Engineering Computing and Programming with

MATLAB 2017

Huei-Huang Lee

Contents

List of Tables

Preface

Use of the Book

This book is developed mainly for junior undergraduate engineering students who have no programming experience, completely new to MATLAB. It may be used in courses such as Computers in Engineering, or others that use MATLAB as a software platform. It also can be used as a self-study book for learning MATLAB.

Features of the Book

Case studies and examples are the core of the book. We believe that the best way to learn MATLAB is to study programs written by experienced programmers and that these example programs determine the quality of a book. The examples in this book are carefully designed to teach students MATLAB programming as well as to inspire their problem-solving potentials. Many are designed to solve a class of problems, instead of a particular problem.

Learning by doing is the teaching approach: students are guided to tackle a problem using MATLAB statements first and then the statements are explained line by line. If the meanings of a statement are obvious, further explanation may not be necessary at all. We believe that this is the most **efficient** and **pain-free** way of learning MATLAB. This approach, together with the extensive use of ordered **textboxes**, **figures**, and **tables**, largely reduces the size of the book, while providing desirable **readability** and **comprehensiveness**.

Junior-college-level engineering problems are used. **Background knowledge** for these engineering problems are illustrated as thoroughly as possible.

Reference materials outside the book can be found mostly either in the **MALAB on-line help** or in **Wikipedia**, their exact locations always pointed out. The idea is that the reference materials should be immediately available for the students; they don't need to go to the library searching for reference books or journal papers. **Cross references** inside the book always come with the **page numbers** to facilitate the reading.

To Students: How to Work with the Book

Chapter 1 introduces MATLAB programming environment and overviews MATLAB functionalities. Chapters 2-9 discuss basic MATLAB functionalities in a progressive and comprehensive way. Chapters 10-13 give advanced topics that are useful in the college years. Each chapter consists of sections, each covering a topic and providing one or more examples. Related MATLAB functions are tabulated at the end of a section. Additional exercise problems are appended at the end of Chapters 2-9.

Examples in a section are presented in a consistent way. An example is usually described first, followed by a MATLAB script. In many cases, the explanation is not even needed, since, as you type and execute the commands, you'll appreciate the concepts that the example intends to convey. Text/graphics output, sometimes input as well, is presented. The rest of the text is to explain the statements of the script.

Although providing all program files (see SDC Publications Website, next page), we hope that you type each statement yourself, since as you type you are acquainting yourself with the MATLAB language. Often, you mistype and take lots of time to fix the mistakes. However, fixing errors is a precious experience of the learning process. Further, whenever possible, execute the statements one at a time (If necessary, using debugging mode; see Section 1.11) and watch the outcome of each statement, i.e., the text/graphic output on the screen.

One of the objectives of this book is to familiarize you with MATLAB on-line documentation, to ensure your continuing growth of MATLAB programing techniques even after the completion of this book. Therefore, consult the MATLAB on-line documentation as often as possible. In this book, we expect you to look up the on-line documentation yourself, whenever a new function is encountered.

To Instructors: How I Use the Book

The book is designed to be like a workbook. Each chapter should be able to be completed in 1-2 weeks. Each week in the classroom, I briefly introduce the materials and, after the classroom hours, let the students work with the book themselves on computers. I also create a weekly on-line discussion forum and let the students post their results, questions, comments, extra work, and so forth. I rate each student's post with a score of 0-5. The accumulated score becomes the grade for the week. I also participate in the discussion and record students' questions that are worth further illustration in the classroom. This teaching model has proved to be efficient and my students love it.

SDC Publications Website

There are no supporting files needed for this book. You type, save, and run each example program, and observe the outcome of the program. However, if you really hate typing, all the program files are available for free download from either the SDC publications website or the author's webpage.

Author's Webpage

A dedicate webpage for this book is maintained by the author:
http://myweb.ncku.edu.tw/~hhlee/Myweb_at_NCKU/MATLAB2017.html

Notations

To efficiently present the material, the writing of this book is not always done in a traditional format. Chapters and sections are numbered in a traditional way, e.g., Chapter 1, Section 1.2, etc. Textboxes in a section are ordered with numbers enclosed by square brackets (e.g., [3]). We may refer to the third textbox in Section 1.2 as "1.2[3]." When referring to a textbox from the same section, we drop the section identifier; for the foregoing example, we simply write "[3]." Equations are numbered in a similar way, except that lower-case letters enclosed by round brackets (parentheses) are used to identify the equations. For example, "1.2(a)" refers to the first equation in Section 1.2. These notations are summarized as follows:

[1], [2], ...	Numbers enclosed by square brackets are used to identify textboxes.
(a), (b), ...	Lower-case letters enclosed by round brackets are used to identify equations.
Reference	References are italicized.
Workspace	Boldface is used to highlight words, specifically MATLAB keywords.
Round-cornered textbox	A round-cornered textbox indicates that mouse or keyboard actions are needed.
Sharp-cornered textbox	A sharp-cornered textbox is for commentary, no mouse or keyboard actions needed.
→	A right arrow means that the next textbox is on the next page.
#	A symbol # means that it is the last textbox of a section.

Huei-Huang Lee

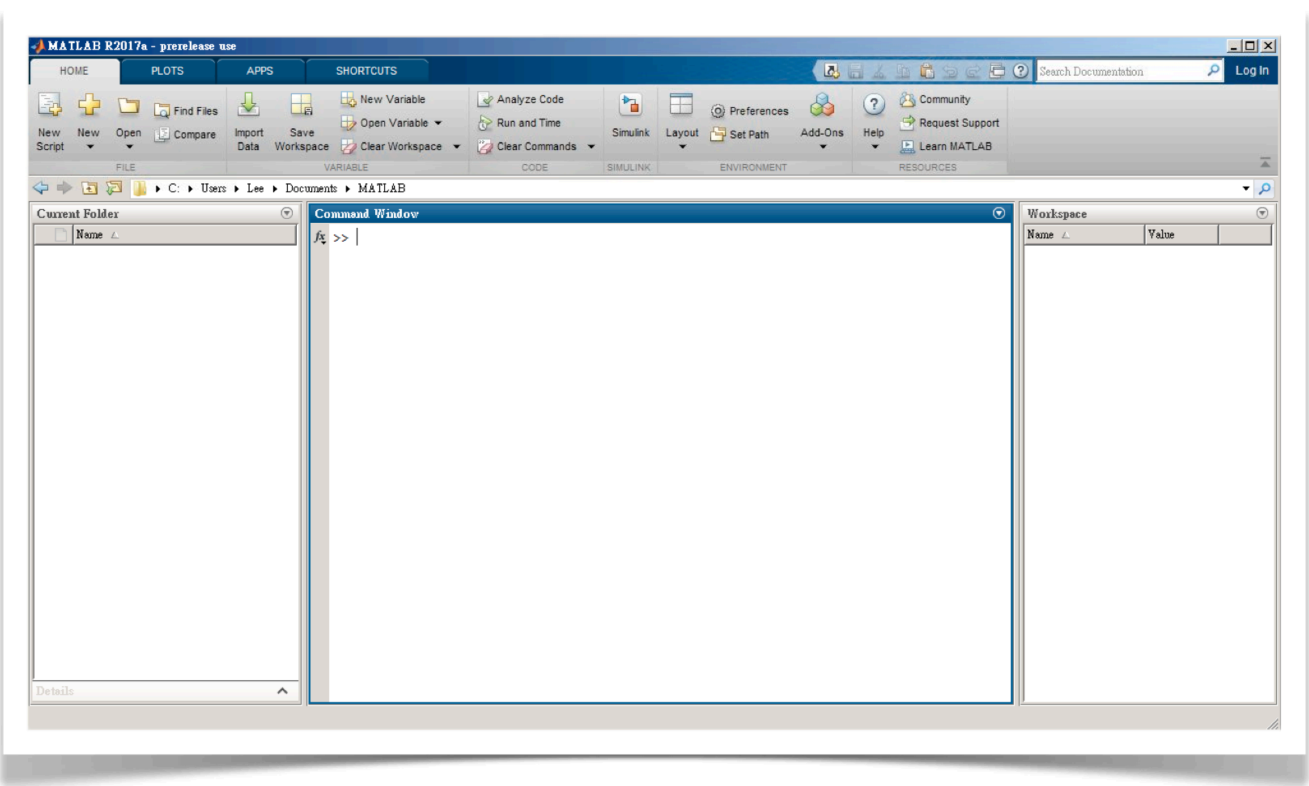
Associate Professor
Department of Engineering Science
National Cheng Kung University, Taiwan
e-mail: hhlee@mail.ncku.edu.tw
webpage: myweb.ncku.edu.tw/~hhlee

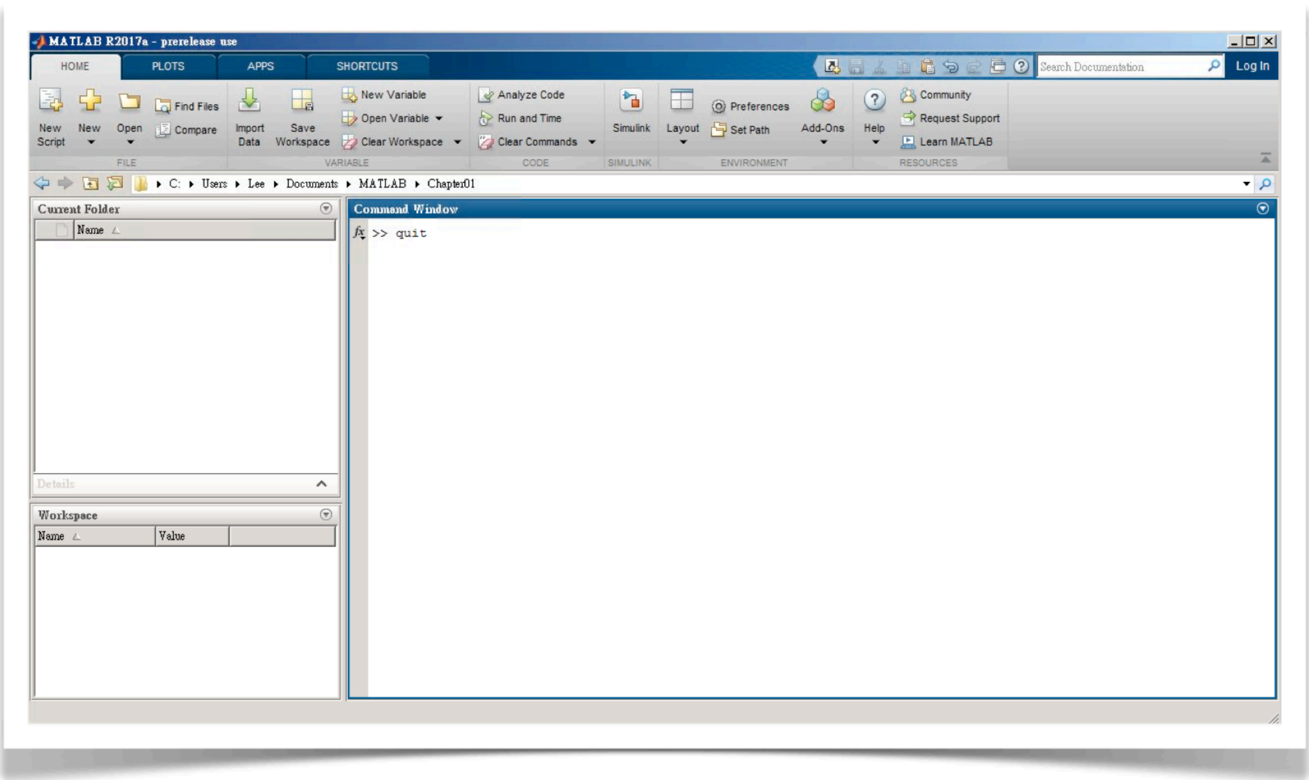
Chapter 1

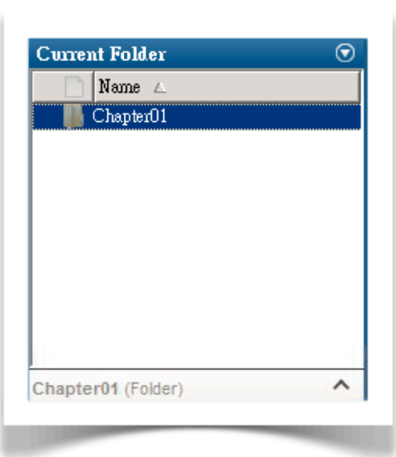
Getting Started, Desktop Environment, and Overview

This chapter introduces MATLAB desktop environment and gives you an overview of the MATLAB functionalities. Each section either introduces a topic that will be detailed in the future chapters or presents features of the MATLAB desktop environment that will be useful throughout the book. The introduction of the topics is to provide a overall, yet incomplete picture of MATLAB. If you feel it is difficult to fully comprehend, don't worry, we'll give you the details in the future chapters. Now, fasten your seat belt and enjoy the learning experience...

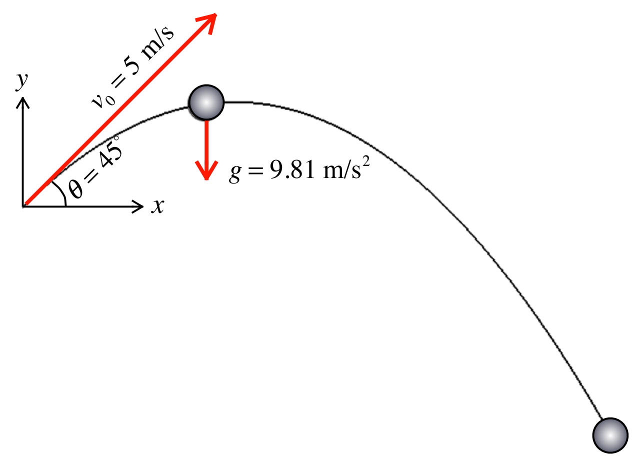
1.1 Start and Quit MATLAB




MATLAB R2017a - prerelease use
HOME
PLOTS
APPS
SHORTCUTS
Search Documentation
Log In
New Script
New
Open
Find Files
Compare
FILE
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
VARIABLE
Analyze Code
Run and Time
Clear Commands
CODE
Simulink
SIMULINK
Layout
Preferences
Set Path
Add-Ons
ENVIRONMENT
Help
Community
Request Support
Learn MATLAB
RESOURCES
C: ▸ Users ▸ Lee ▸ Documents ▸ MATLAB ▸ Chapter01
Current Folder
Name
Details
Workspace
Name
Value
Command Window
>> quit


Current Folder
Name
Chapter01
Chapter01 (Folder)

1.2 Entering Commands



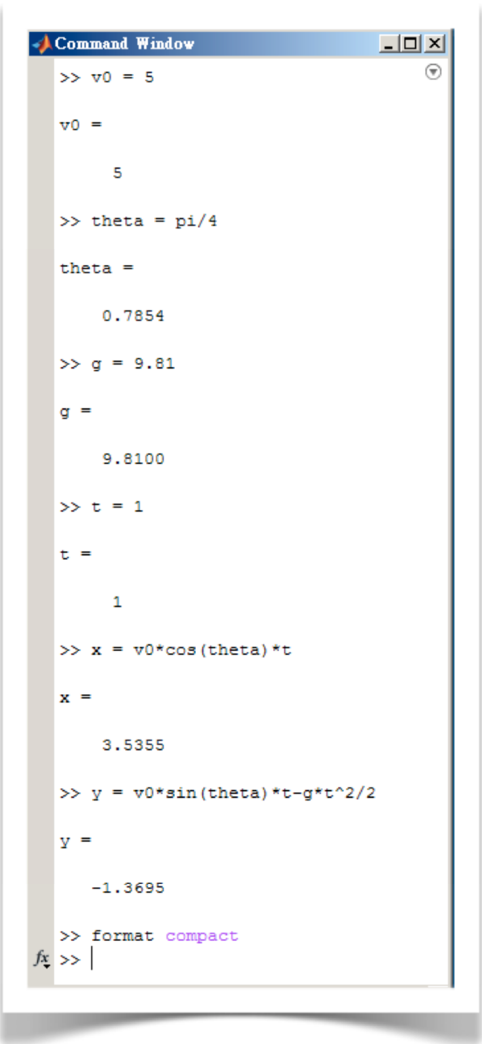
[7] Enter the following commands one after another and watch the text output on your **Command Window** (see [8], next page, for the **Command Window** output).

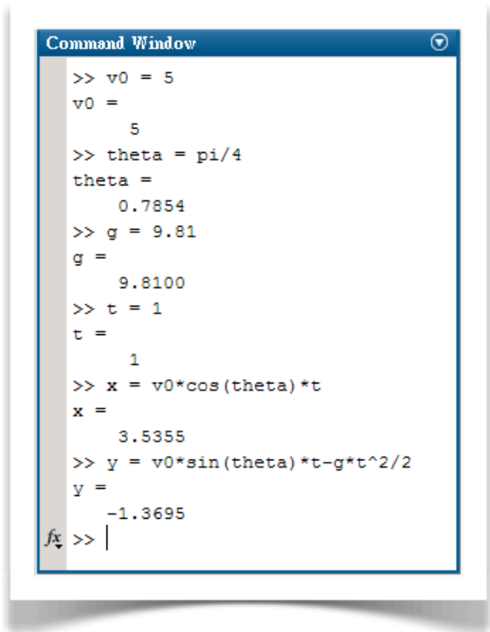
```
>> v0 = 5
>> theta = pi/4
>> g = 9.81
>> t = 1
>> x = v0*cos(theta)*t
>> y = v0*sin(theta)*t-g*t^2/2
```

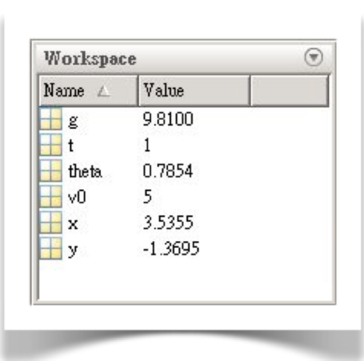
In the first command, a variable `v0` is created in the **Workspace** and a value `5` is assigned to the variable.

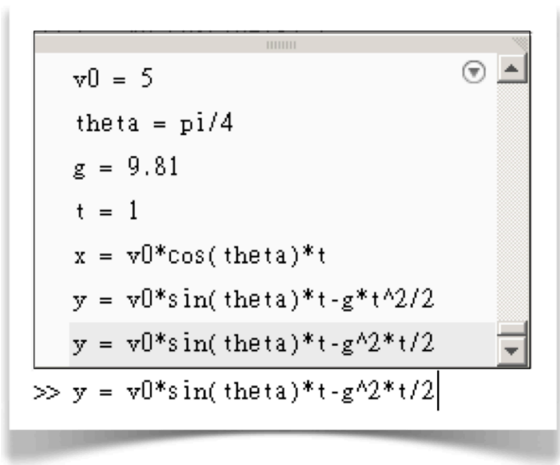
In the second command, another variable `theta` is created and the value `pi` (which is a built-in constant with a value of 3.141592653589793) is divided by 4 and assigned to the variable. The third and fourth commands are self-explained.

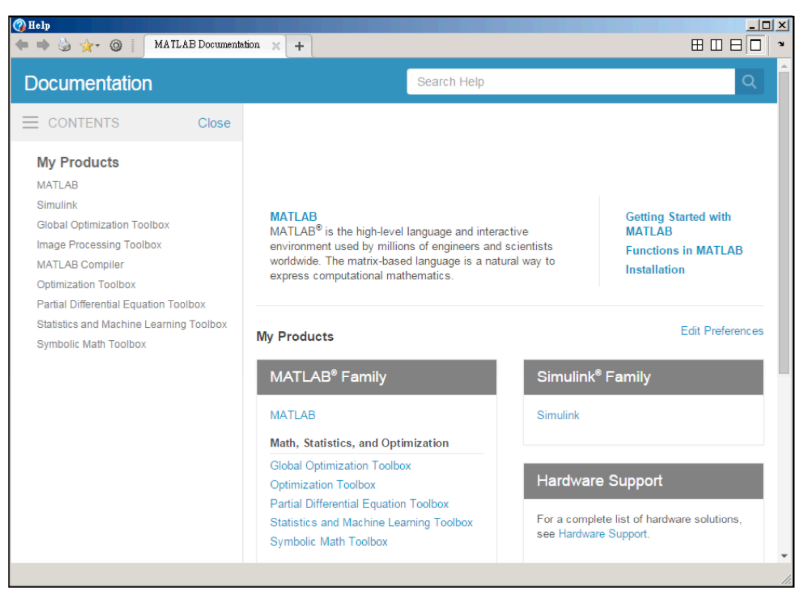
In the fifth command, x-position is calculated using Eq. (a). In the sixth command, y-position is calculated using Eq. (b). Note that `cos` and `sin` are MATLAB built-in functions; each takes radius as angular unit. →


Command Window
>> v0 = 5
v0 =
5
>> theta = pi/4
theta =
0.7854
>> g = 9.81
g =
9.8100
>> t = 1
t =
1
>> x = v0*cos(theta)*t
x =
3.5355
>> y = v0*sin(theta)*t-g*t^2/2
y =
-1.3695
>> format compact
fx >>


Command Window
>> v0 = 5
v0 =
5
>> theta = pi/4
theta =
0.7854
>> g = 9.81
g =
9.8100
>> t = 1
t =
1
>> x = v0*cos(theta)*t
x =
3.5355
>> y = v0*sin(theta)*t-g*t^2/2
y =
-1.3695
fx >>


Workspace
Name
Value
g 9.8100
t 1
theta 0.7854
v0 5
x 3.5355
y -1.3695


v0 = 5
theta = pi/4
g = 9.81
t = 1
x = v0*cos(theta)*t
y = v0*sin(theta)*t-g*t^2/2
y = v0*sin(theta)*t-g^2*t/2
>> y = v0*sin(theta)*t-g^2*t/2


Help
MATLAB Documentation
Documentation
Search Help
CONTENTS
Close
My Products
MATLAB
Simulink
Global Optimization Toolbox
Image Processing Toolbox
MATLAB Compiler
Optimization Toolbox
Partial Differential Equation Toolbox
Statistics and Machine Learning Toolbox
Symbolic Math Toolbox
MATLAB
MATLAB® is the high-level language and interactive environment used by millions of engineers and scientists worldwide. The matrix-based language is a natural way to express computational mathematics.
Getting Started with MATLAB
Functions in MATLAB
Installation
My Products
Edit Preferences
MATLAB® Family
Simulink® Family
MATLAB
Simulink
Math, Statistics, and Optimization
Global Optimization Toolbox
Optimization Toolbox
Partial Differential Equation Toolbox
Statistics and Machine Learning Toolbox
Symbolic Math Toolbox
Hardware Support
For a complete list of hardware solutions, see Hardware Support.

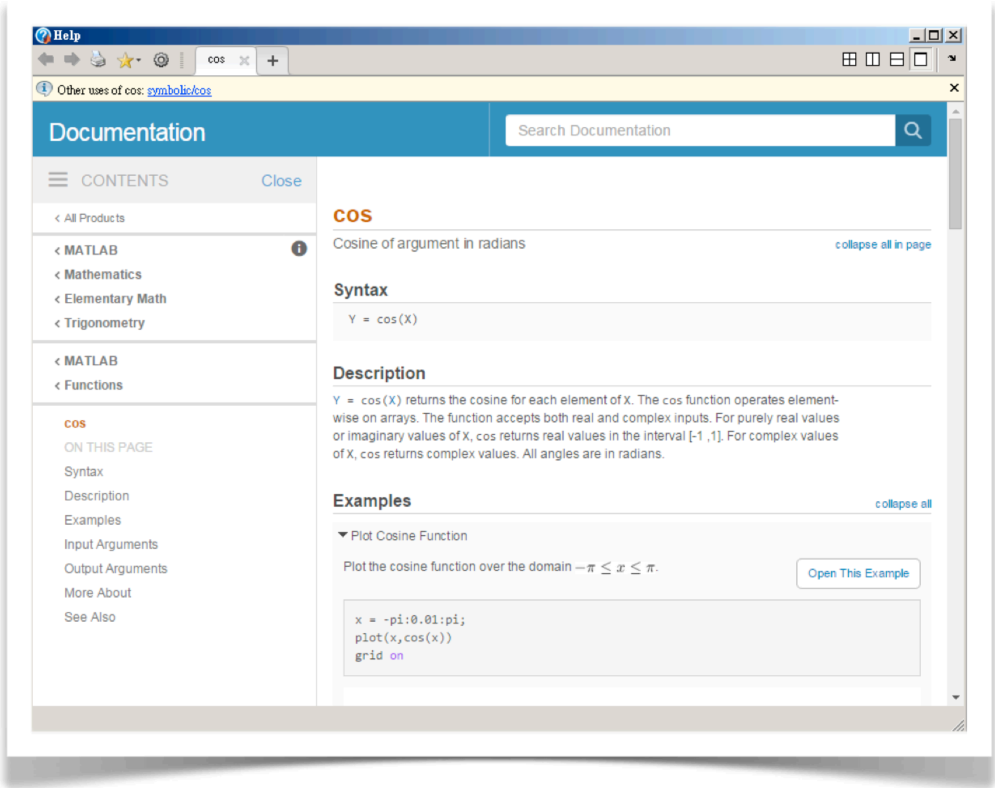


Table 1.2 Rules for Variable Names

Variable names are examples of identifier names, which include **variable names** and **function names**. The following are he rules for an identifier name:

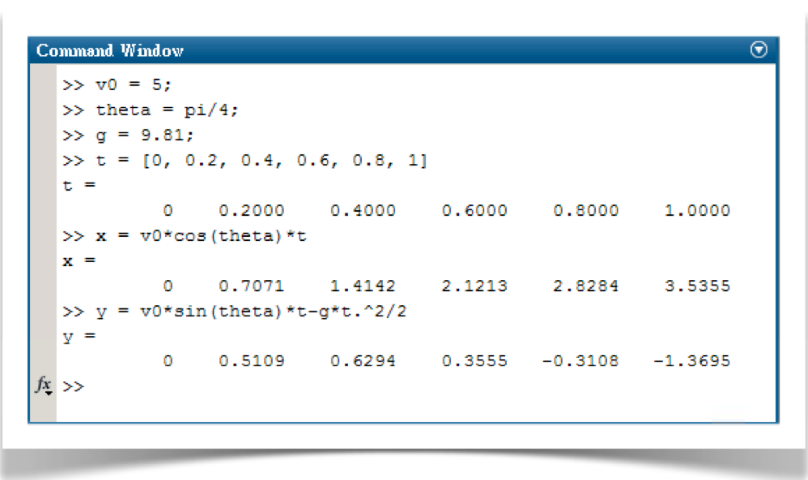
- A name contains only letters (A, a, B, b, ... Z, z), digits (0, 1, 2, ..., 9), and underscore (_).
- A name must begin with a letter.
- The maximum length of a name is 63. (*Type* >> `namelengthmax` *to obtain this number.*)
- A name is case-sensitive; i.e., there is a difference between uppercase and lowercase letters.
- It is legal, but not recommended, to use a built-in function name as a user-defined name. In that case, the built-in function is temporarily "invisible."
- The following MATLAB keywords cannot be used as a name: (*Type* >> `iskeyword` *to obtain this list.*)

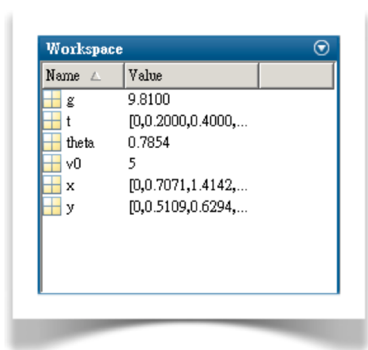
break	case	catch	classdef	continue	else	elseif
end	for	function	global	if	otherwise	parfor
persistent	return	spmd	switch	try	while	

1.3 Array Expressions

[2] The following commands calculate the positions of the ball at 0 sec, 0.2 sec, 0.4 sec, 0.6 sec, 0.8 sec, and 1.0 sec. Your **Command Window** and **Workspace Window** should look like [3] and [4], respectively.

```
>> v0 = 5;
>> theta = pi/4;
>> g = 9.81;
>> t = [0, 0.2, 0.4, 0.6, 0.8, 1]
>> x = v0*cos(theta)*t
>> y = v0*sin(theta)*t-g*t.^2/2
```

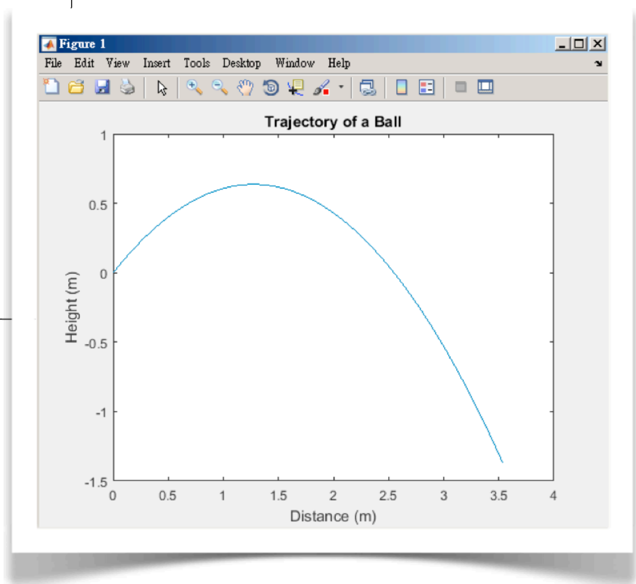




1.4 Data Visualization: Line Plots

[2] Following commands draw a trajectory of a ball (see [3]).

```
>> clear, clc
>> v0 = 5; theta = pi/4; g = 9.81;
>> t = 0:0.02:1;
>> x = v0*cos(theta)*t;
>> y = v0*sin(theta)*t-g*t.^2/2;
>> plot(x, y)
>> title('Trajectory of a Ball')
>> xlabel('Distance (m)')
>> ylabel('Height (m)')
```



```
>> t = 0:0.02:1
t =
  Columns 1 through 5
         0    0.0200    0.0400    0.0600    0.0800
  Columns 6 through 10
    0.1000    0.1200    0.1400    0.1600    0.1800
  Columns 11 through 15
    0.2000    0.2200    0.2400    0.2600    0.2800
  Columns 16 through 20
    0.3000    0.3200    0.3400    0.3600    0.3800
  Columns 21 through 25
    0.4000    0.4200    0.4400    0.4600    0.4800
  Columns 26 through 30
    0.5000    0.5200    0.5400    0.5600    0.5800
  Columns 31 through 35
    0.6000    0.6200    0.6400    0.6600    0.6800
  Columns 36 through 40
    0.7000    0.7200    0.7400    0.7600    0.7800
  Columns 41 through 45
    0.8000    0.8200    0.8400    0.8600    0.8800
  Columns 46 through 50
    0.9000    0.9200    0.9400    0.9600    0.9800
  Column 51
    1.0000
```

3-D Line Plots

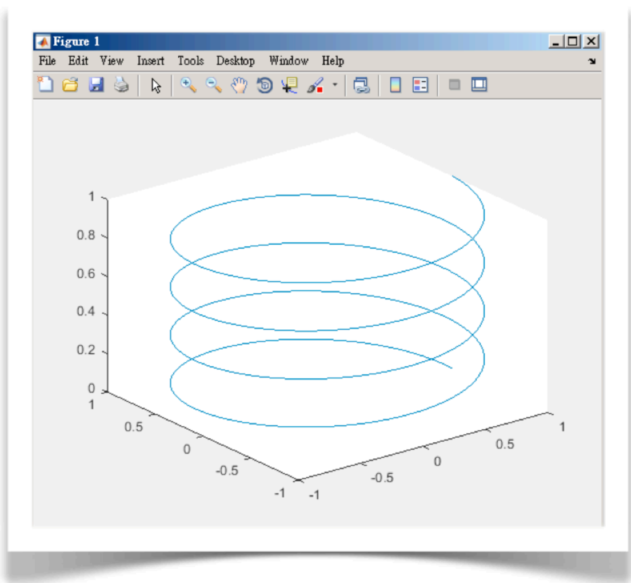
[10] The function `plot3(x,y,z)` produces a curve in the 3-D space. The curve is produced by connecting the following data points with straight line segments:

$$(x(i),\ y(i),\ z(i)),\ i = 1,\ 2,\ \dots n$$

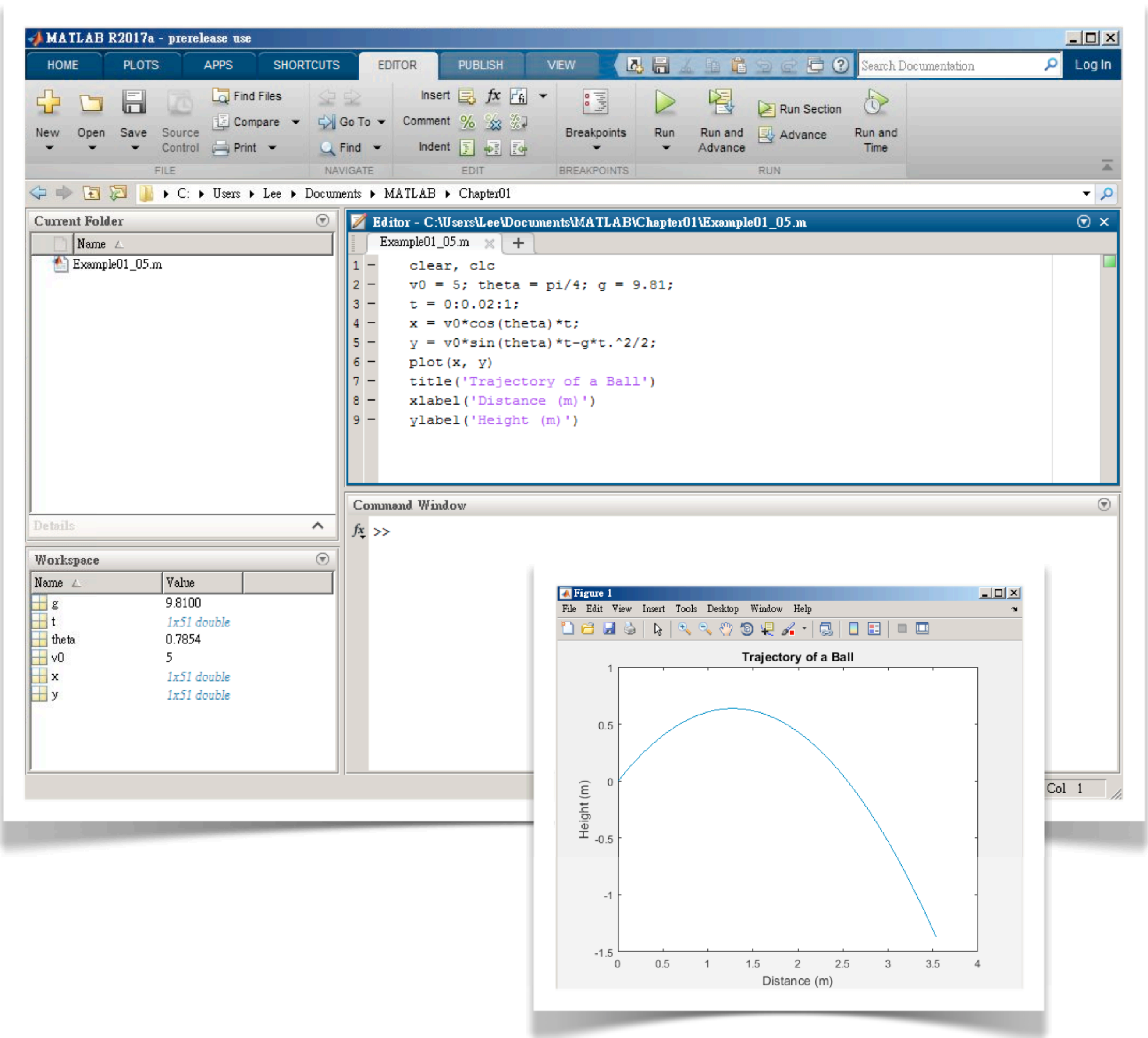
As an example, following commands draw a spiral curve as shown in [11].

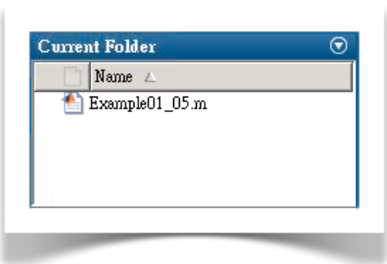
```
>> clear, clc, close
>> theta = 0:pi/100:8*pi;
>> x = cos(theta);
>> y = sin(theta);
>> z = theta/(8*pi);
>> plot3(x, y, z)
```

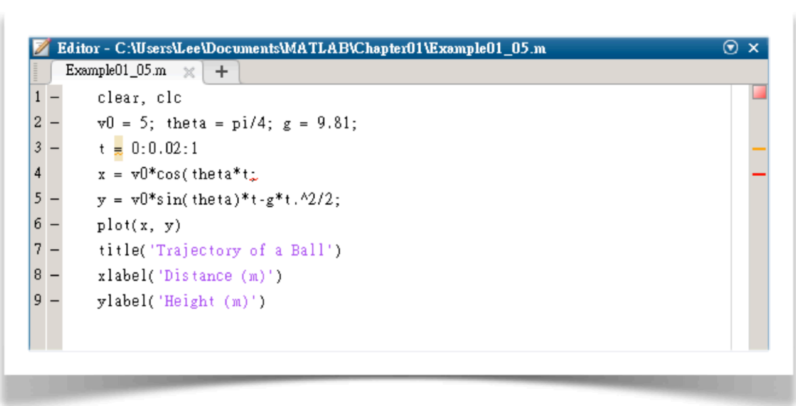
Note that the `close` command in the first line closes the "current" **Figure Window** ([3], page 19).



1.5 MATLAB Scripts




Current Folder
Name
Example01_05.m


Editor - C:\Users\Lee\Documents\MATLAB\Chapter01\Example01_05.m
Example01_05.m
clear, clc
v0 = 5; theta = pi/4; g = 9.81;
t = 0:0.02:1
x = v0*cos(theta*t;
y = v0*sin(theta)*t-g*t.^2/2;
plot(x, y)
title('Trajectory of a Ball')
xlabel('Distance (m)')
ylabel('Height (m)')

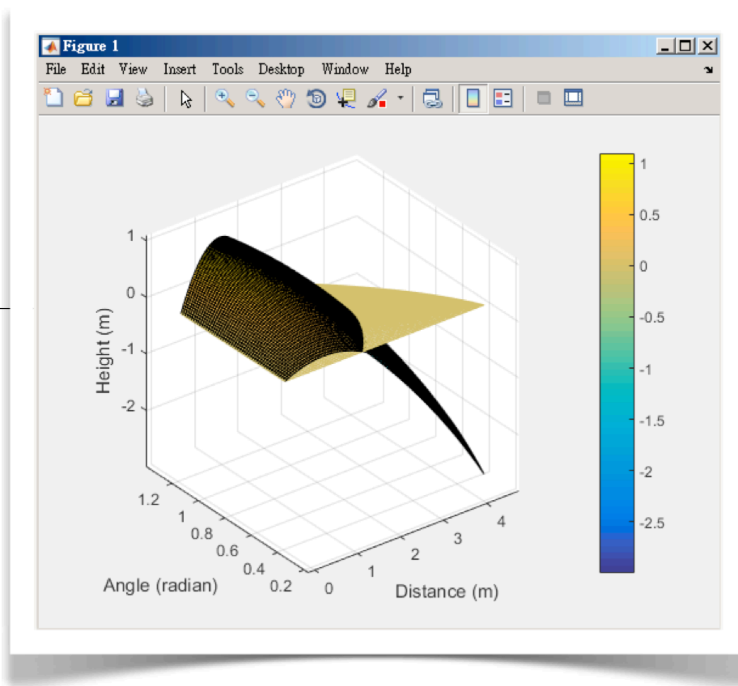
```
% This is a program to calculate
% and plot the trajectory of a ball

clear, clc % clear Workspace and Command Window

% Initial speed v0, m/s
% Elevation angle theta, radians
% Gravitational acceleration g, m/s^2
v0 = 5; theta = pi/4; g = 9.81;

t = 0:0.02:1; % The time ranges from 0 sec to 1 sec
x = v0*cos(theta)*t; % x-coordinates
y = v0*sin(theta)*t-g*t.^2/2; % y-coordinates

% Plot the trajectory
plot(x, y)
title('Trajectory of a Ball')
xlabel('Distance (m)')
ylabel('Height (m)')
```

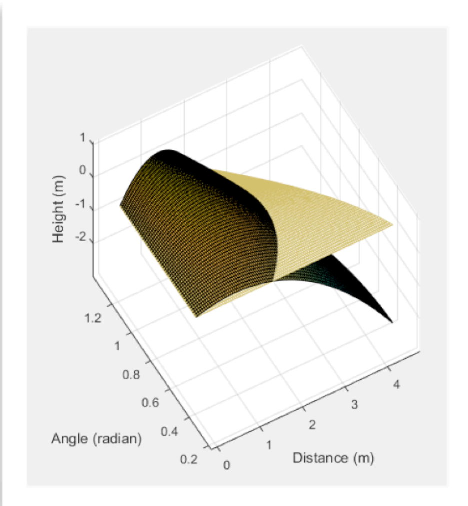
1.6 Data Visualization: Surface Plots

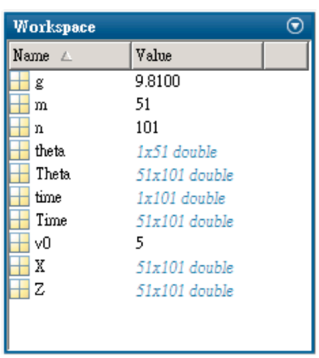
Example01_06.m: Trajectory Surface

[2] Create a new script (1.5[2], page 22), type the following commands (the number before each line is added by the author and is not part of the command), save as Example01_06.m (1.5[6]), and run the script (1.5[8]). The graphic output is shown in [3-5]. We'll explain these lines in [9-19].

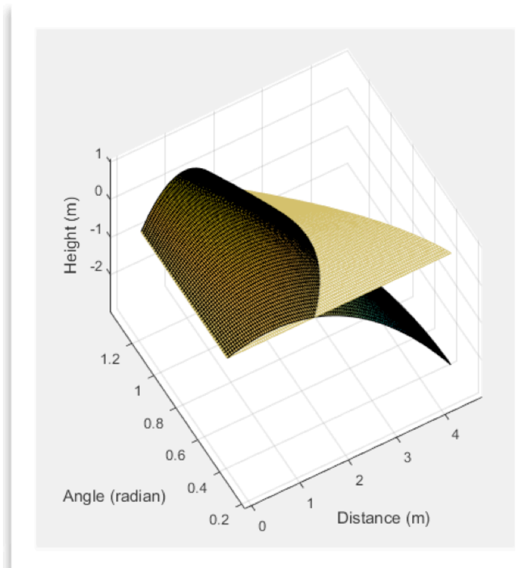
```
clear, clc, close all
v0 = 5; g = 9.81;
time = 0:0.01:1; n = length(time);
theta = pi/8:pi/200:3*pi/8; m = length(theta);
Time = repmat(time, m, 1);
Theta = repmat(theta', 1, n);
X = v0*cos(Theta).*Time;
Z = v0*sin(Theta).*Time-g*Time.^2/2;
surf(X, Theta, Z)
hold on
Z = zeros(m, n);
mesh(X, Theta, Z)
xlabel('Distance (m)')
ylabel('Angle (radian)')
zlabel('Height (m)')
colorbar
axis vis3d
```




Height (m)
1
0
-1
-2
1.2
1
0.8
0.6
0.4
0.2
Angle (radian)
0
1
2
3
4
Distance (m)


Workspace
Name
Value
g 9.8100
m 51
n 101
theta 1x51 double
Theta 51x101 double
time 1x101 double
Time 51x101 double
v0 5
X 51x101 double
Z 51x101 double

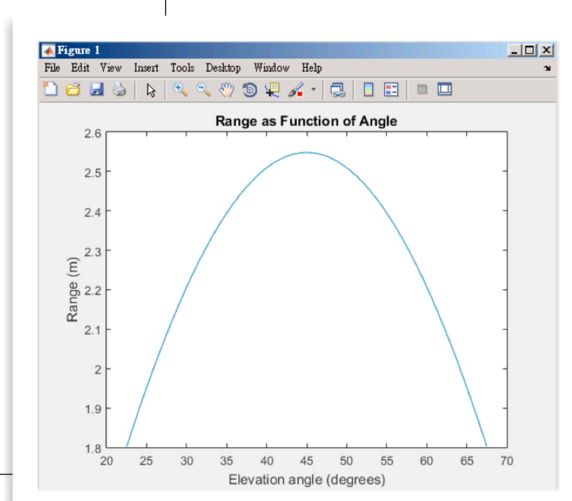
1.7 Symbolic Mathematics

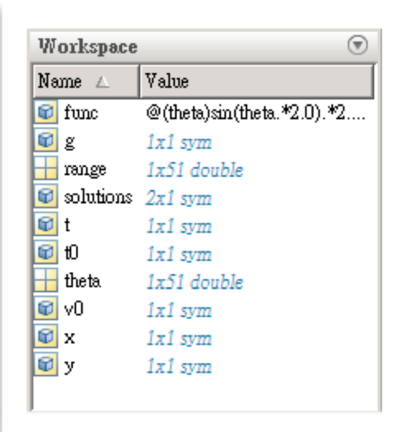


Example01_07.m: Ball-Throwing Ranges

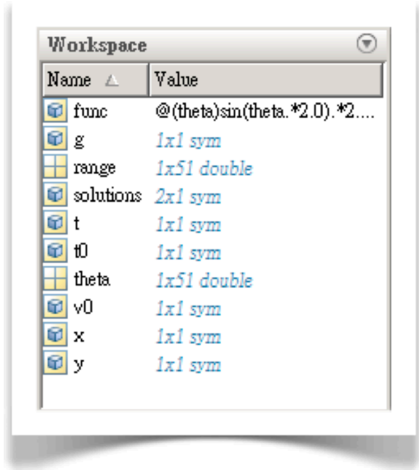
[3] Create a new script, type the following commands, save as Example01_07.m, and run the script.

```
clear, clc, close all
syms v0 theta g t
x = v0*cos(theta)*t
y = v0*sin(theta)*t-g*t^2/2
solutions = solve(y, t)
t0 = solutions(2)
range = subs(x, t, t0)
range = simplify(range)
range = subs(range, [v0, g], [5, 9.81])
func = matlabFunction(range)
theta = [pi/8:pi/200:3*pi/8];
range = func(theta);
plot(theta*180/pi, range)
title('Range as Function of Angle')
xlabel('Elevation angle (degrees)')
ylabel('Range (m)')
```

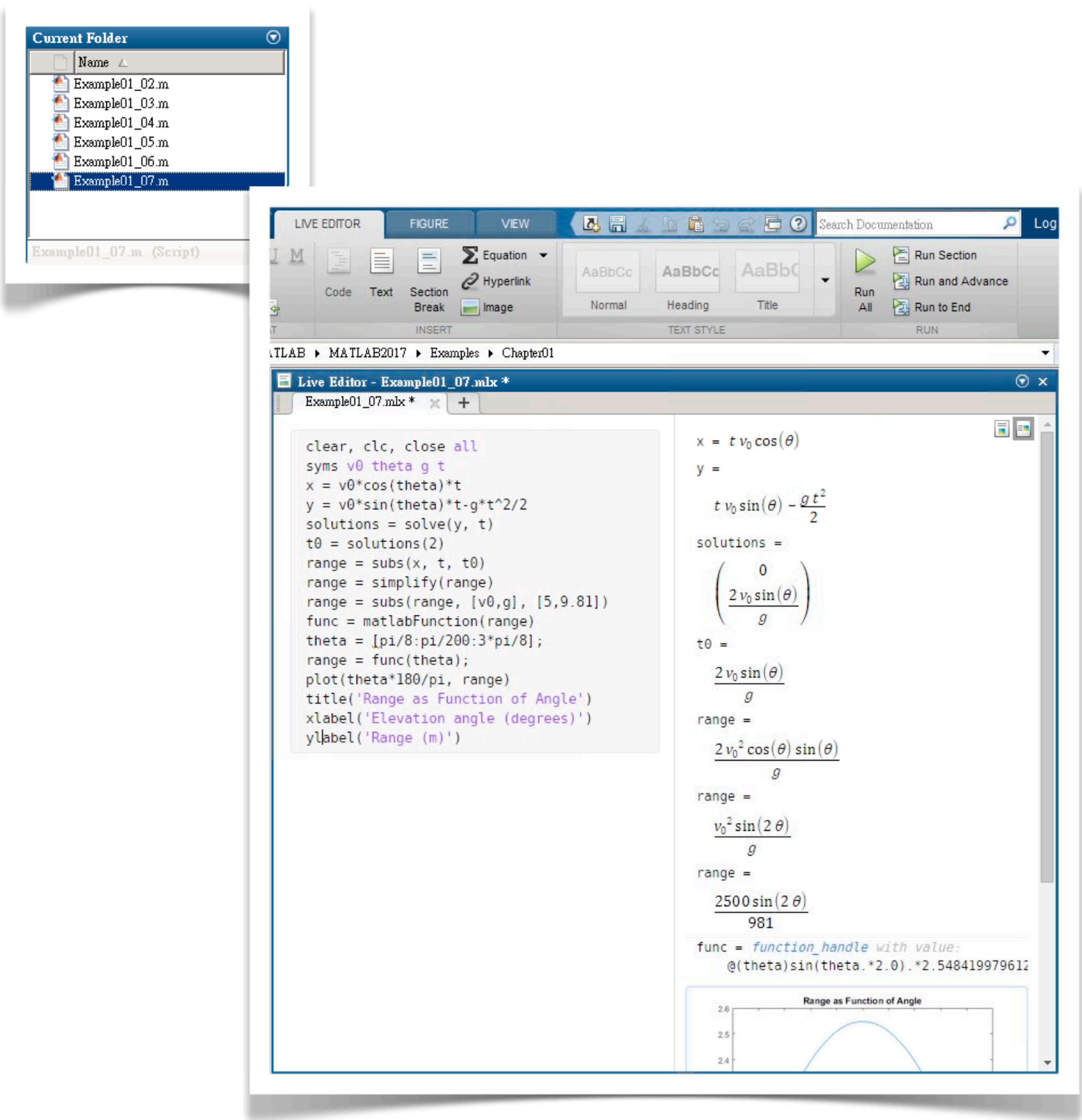





```
x =
t*v0*cos(theta)
y =
t*v0*sin(theta) - (g*t^2)/2
solutions =
                   0
 (2*v0*sin(theta))/g
t0 =
(2*v0*sin(theta))/g
range =
(2*v0^2*cos(theta)*sin(theta))/g
range =
(v0^2*sin(2*theta))/g
range =
(2500*sin(2*theta))/981
func =
  function_handle with value:
    @(theta)sin(theta.*2.0).*2.54841997961264
```


Workspace
Name
Value
func
@(theta)sin(theta.*2.0).*2....
g
1x1 sym
range
1x51 double
solutions
2x1 sym
t
1x1 sym
t0
1x1 sym
theta
1x51 double
v0
1x1 sym
x
1x1 sym
y
1x1 sym

1.8 Live Script



Range of a Ball

This program symbolically calculates the range of a ball throwing with an initial speed of v_0 and an elevation angle of θ.

```
clear, clc, close all % Housekeeping
```

Trajectory of the ball, (x, y)

Initial speed v_0, elevation angle θ, gravitational acceleration g, time t

$$x = (v_0\cos\theta)t\,,\; y = (v_0\sin\theta)t - \frac{1}{2}gt^2$$

```
syms v0 theta g t
x = v0*cos(theta)*t
y = v0*sin(theta)*t-g*t^2/2
```

Time to hit the ground, t_0

Solve the equation $y = 0$ for t

```
solutions = solve(y, t)
t0 = solutions(2) % The first solution is 0
```

Range of the ball, $R = x(t_0)$

```
range = subs(x, t, t0) % range = x(t0)
range = simplify(range)
range = subs(range, [v0, g], [5, 9.81])
```

Numerical values $R(\theta)$, $\theta = \dfrac{\pi}{8}$ to $\dfrac{3\pi}{8}$

```
func = matlabFunction(range)
theta = [pi/8:pi/200:3*pi/8];
range = func(theta);
```

R-versus-θ plot

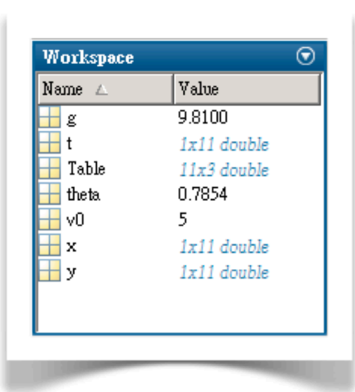
```
plot(theta*180/pi, range)
title('Range as Function of Angle')
xlabel('Elevation angle (degrees)')
ylabel('Range (m)')
```

1.9 Screen Text Input/Output

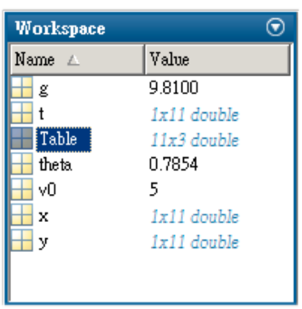
Example01_09.m: Trajectory Table

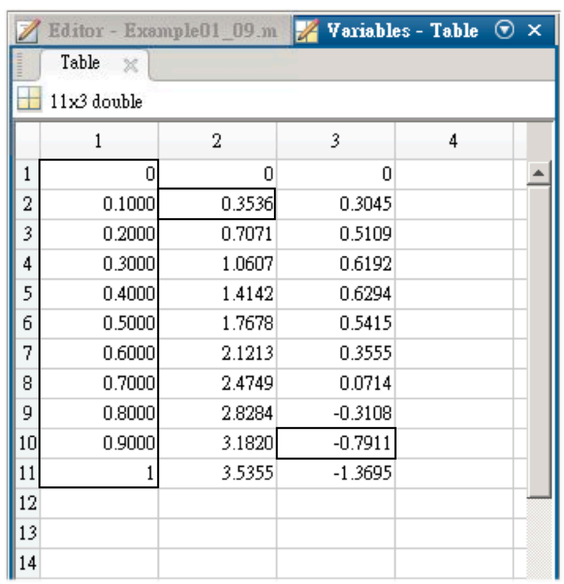
[2] Create a new file, type the following commands, save as Example01_09.m, and run the script. Enter 5 for the initial velocity and 45 for the elevation angle (see [3]).

```
clear
v0 = input('Enter initial speed (m/s): ');
theta = input('Enter elevation angle (degrees): ');
theta = theta*pi/180;
g = 9.81;
t = 0:0.1:1;
x = v0*cos(theta)*t;
y = v0*sin(theta)*t-g*t.^2/2;
Table = [t', x', y'];
disp(' ')
disp('  time (s)     x (m)     y (m)')
disp(Table)
```

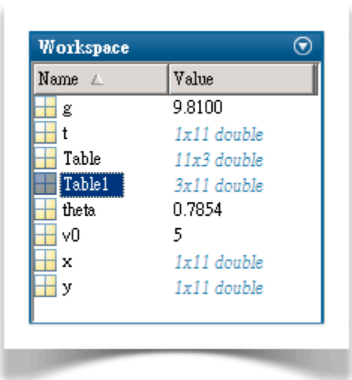


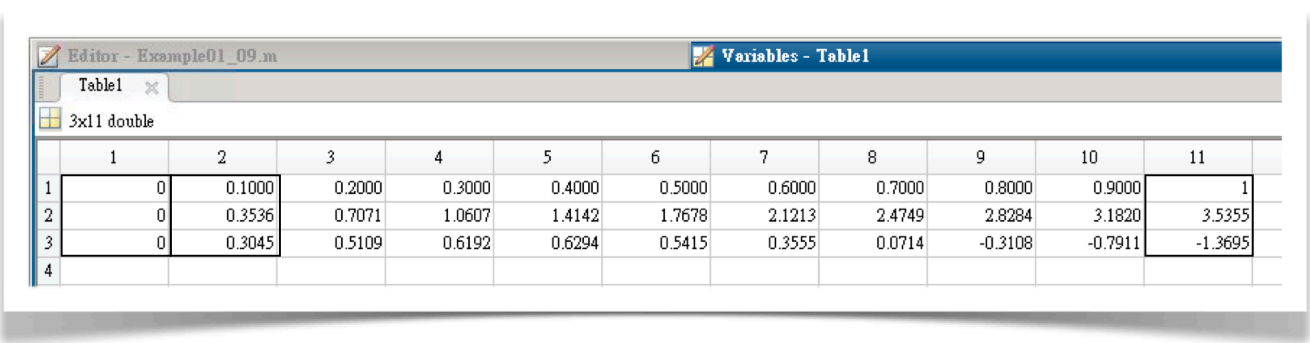
```
time (s)      x (m)      y (m)
     0.0      0.000      0.000
     0.1      0.354      0.305
     0.2      0.707      0.511
     0.3      1.061      0.619
     0.4      1.414      0.629
     0.5      1.768      0.542
     0.6      2.121      0.356
     0.7      2.475      0.071
     0.8      2.828     -0.311
     0.9      3.182     -0.791
     1.0      3.536     -1.369
```





	1	2	3	4
1	0	0	0	
2	0.1000	0.3536	0.3045	
3	0.2000	0.7071	0.5109	
4	0.3000	1.0607	0.6192	
5	0.4000	1.4142	0.6294	
6	0.5000	1.7678	0.5415	
7	0.6000	2.1213	0.3555	
8	0.7000	2.4749	0.0714	
9	0.8000	2.8284	-0.3108	
10	0.9000	3.1820	-0.7911	
11	1	3.5355	-1.3695	
12				
13				
14				





	1	2	3	4	5	6	7	8	9	10	11
1	0	0.1000	0.2000	0.3000	0.4000	0.5000	0.6000	0.7000	0.8000	0.9000	1
2	0	0.3536	0.7071	1.0607	1.4142	1.7678	2.1213	2.4749	2.8284	3.1820	3.5355
3	0	0.3045	0.5109	0.6192	0.6294	0.5415	0.3555	0.0714	-0.3108	-0.7911	-1.3695
4											

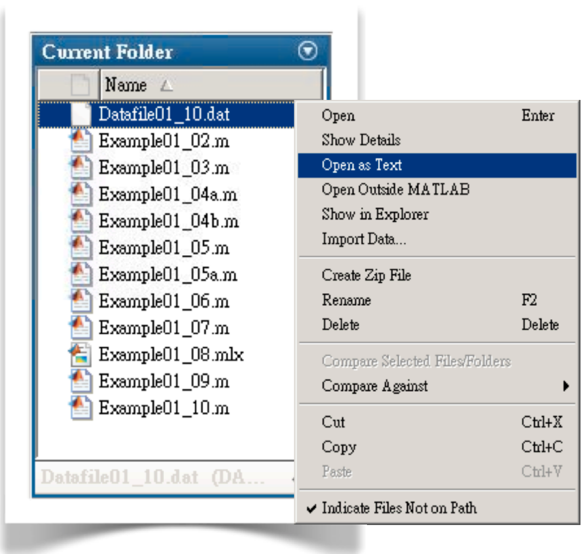
1.10 Text File Input/Output

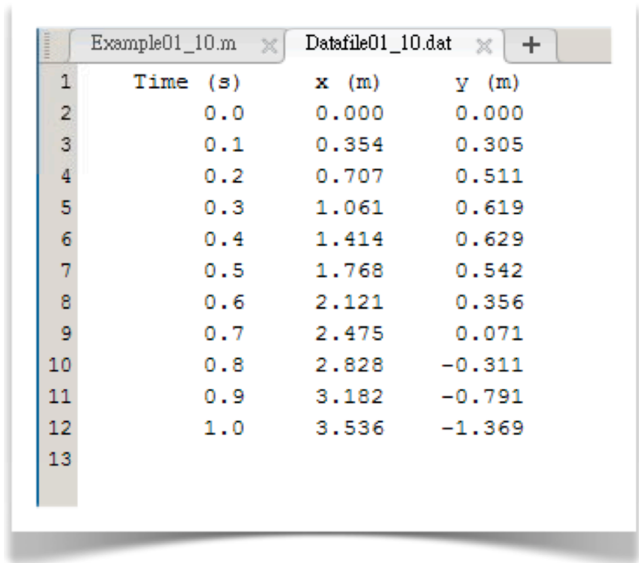
Example01_10.m: Text File I/O

[2] Create a new script, type the following commands, save as Example01_10.m, and run the script. This program demonstrates input/output of a text file.

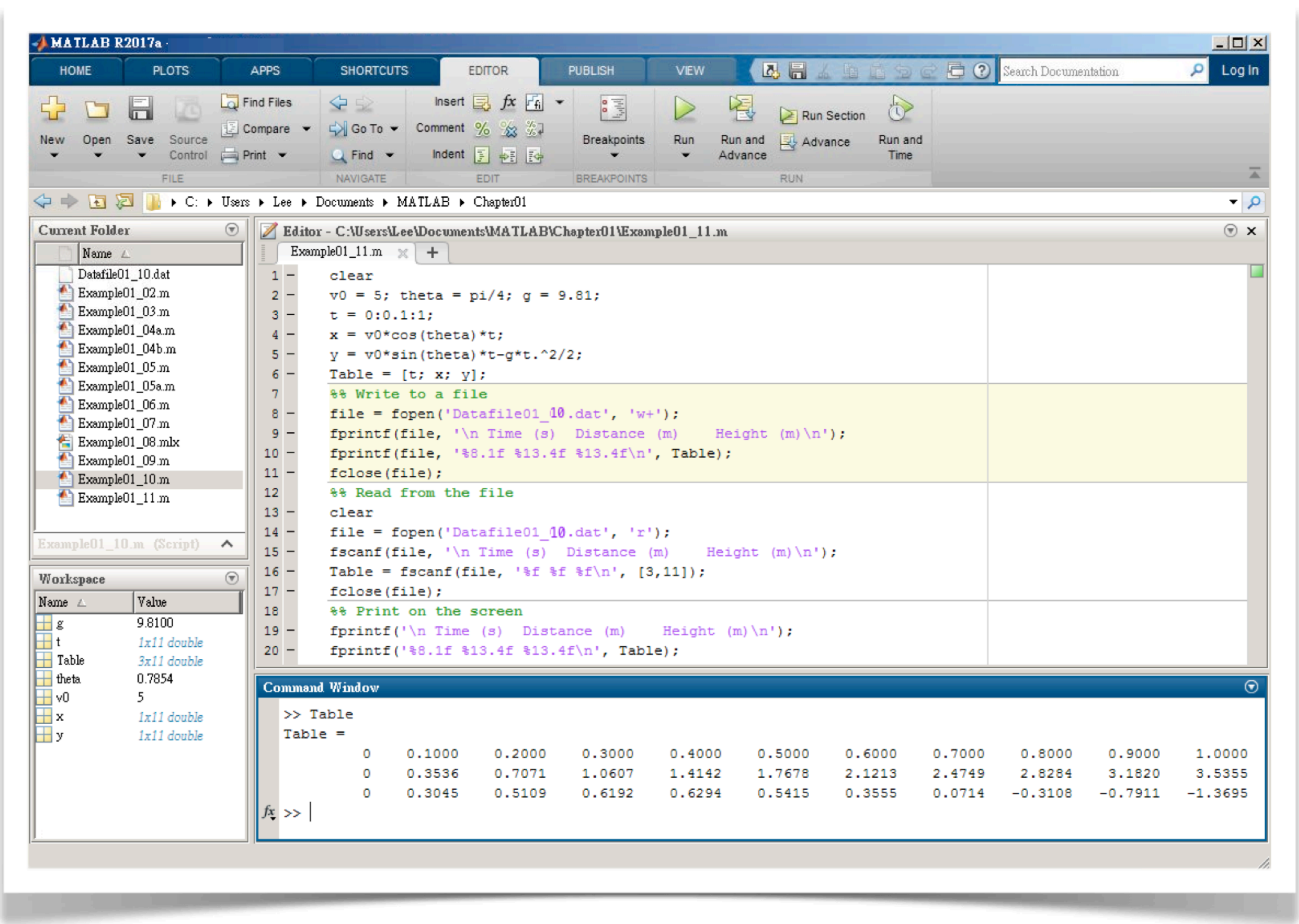
```
clear
v0 = 5; theta = pi/4; g = 9.81;
t = 0:0.1:1;
x = v0*cos(theta)*t;
y = v0*sin(theta)*t-g*t.^2/2;
Table = [t; x; y];
% Write to a file
file = fopen('Datafile01_10.dat', 'w+');
fprintf(file, '  Time (s)     x (m)     y (m)\n');
fprintf(file, '%10.1f %9.3f %9.3f\n', Table);
fclose(file);
% Read from the file
clear
file = fopen('Datafile01_10.dat', 'r');
fscanf(file, '  Time (s)     x (m)     y (m)\n');
Table = fscanf(file, '%f %f %f\n', [3,11]);
fclose(file);
% Print on the screen
fprintf('  Time (s)     x (m)     y (m)\n');
fprintf('%10.1f %9.3f %9.3f\n', Table);
```

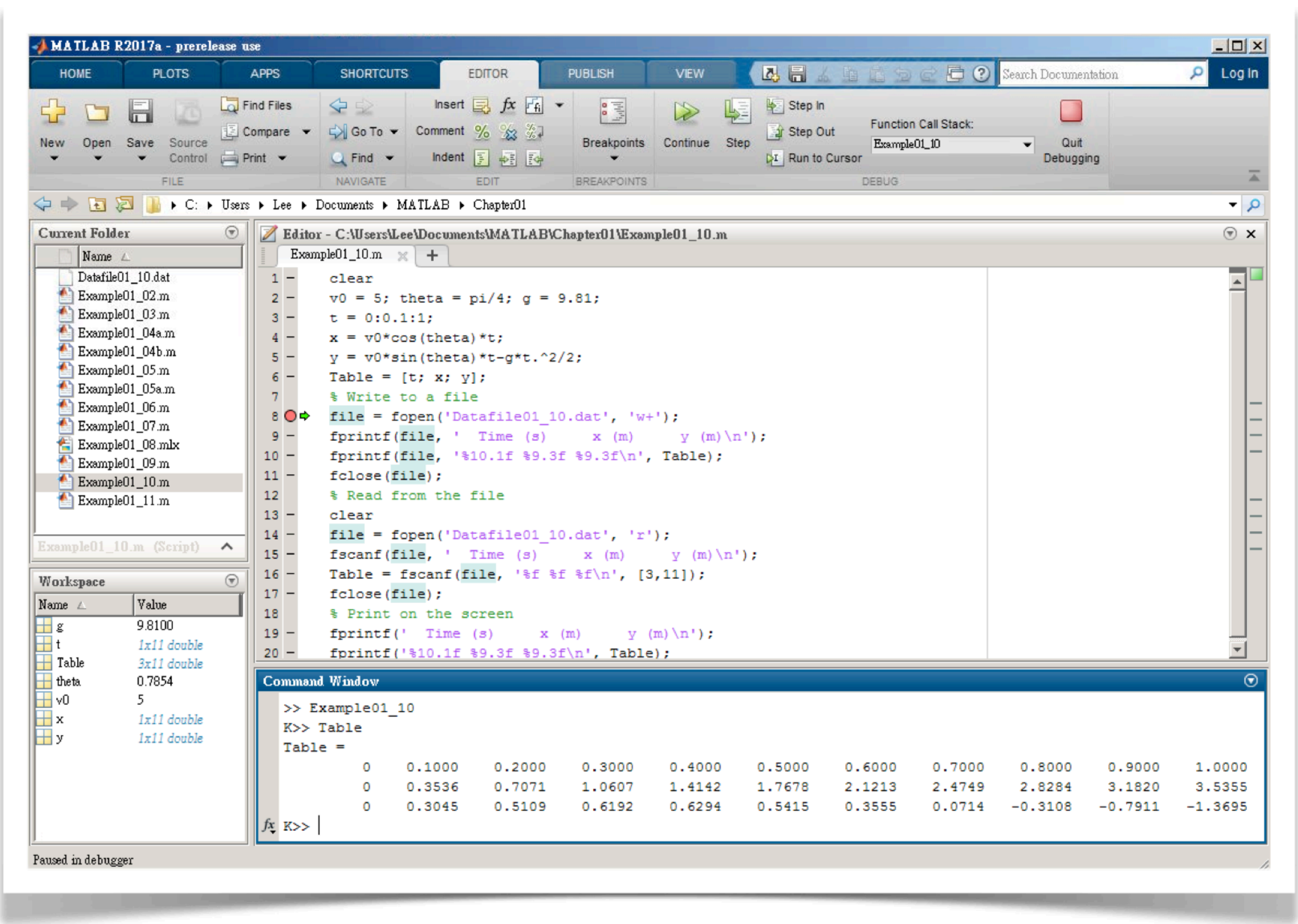
```
>> Example01_10
  Time (s)     x (m)     y (m)
       0.0     0.000     0.000
       0.1     0.354     0.305
       0.2     0.707     0.511
       0.3     1.061     0.619
       0.4     1.414     0.629
       0.5     1.768     0.542
       0.6     2.121     0.356
       0.7     2.475     0.071
       0.8     2.828    -0.311
       0.9     3.182    -0.791
       1.0     3.536    -1.369
>>
```


Current Folder
Name
Datafile01_10.dat
Example01_02.m
Example01_03.m
Example01_04a.m
Example01_04b.m
Example01_05.m
Example01_05a.m
Example01_06.m
Example01_07.m
Example01_08.mlx
Example01_09.m
Example01_10.m
Open Enter
Show Details
Open as Text
Open Outside MATLAB
Show in Explorer
Import Data...
Create Zip File
Rename F2
Delete Delete
Compare Selected Files/Folders
Compare Against
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Indicate Files Not on Path

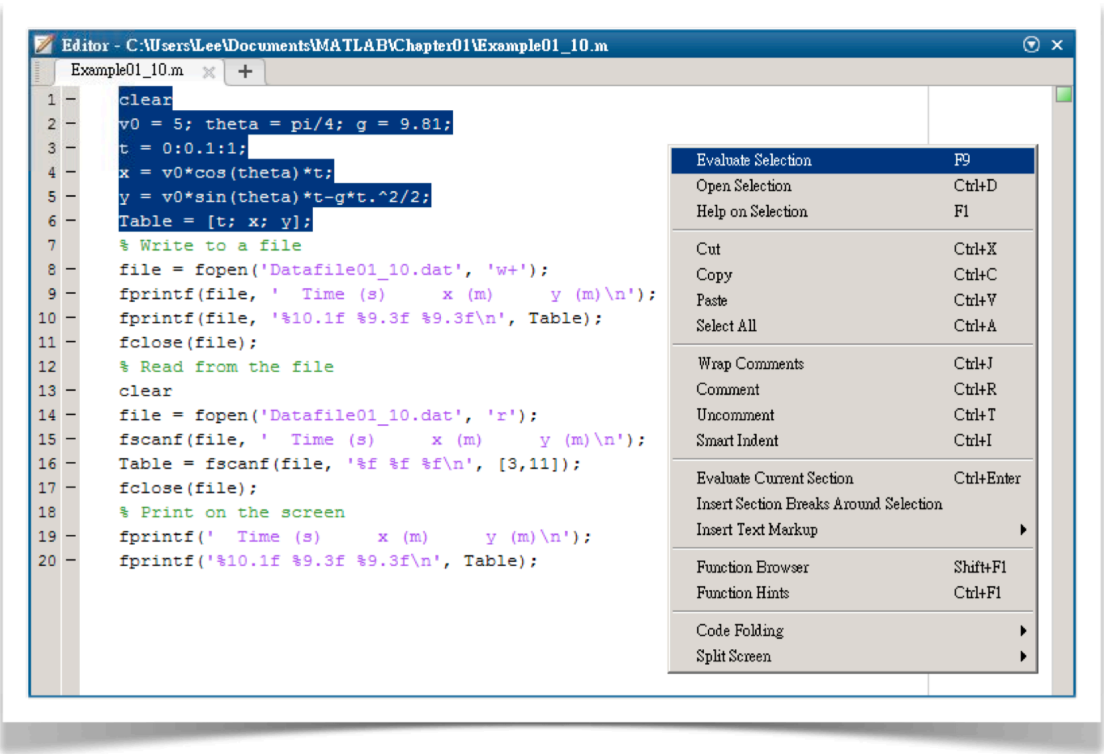

Example01_10.m
Datafile01_10.dat
Time (s) x (m) y (m)
0.0 0.000 0.000
0.1 0.354 0.305
0.2 0.707 0.511
0.3 1.061 0.619
0.4 1.414 0.629
0.5 1.768 0.542
0.6 2.121 0.356
0.7 2.475 0.071
0.8 2.828 -0.311
0.9 3.182 -0.791
1.0 3.536 -1.369

1.11 Debug Your Programs





MATLAB R2017a - prerelease use
Editor - C:\Users\Lee\Documents\MATLAB\Chapter01\Example01_10.m
clear
v0 = 5; theta = pi/4; g = 9.81;
t = 0:0.1:1;
x = v0*cos(theta)*t;
y = v0*sin(theta)*t-g*t.^2/2;
Table = [t; x; y];
% Write to a file
file = fopen('Datafile01_10.dat', 'w+');
fprintf(file, ' Time (s) x (m) y (m)\n');
fprintf(file, '%10.1f %9.3f %9.3f\n', Table);
fclose(file);
% Read from the file
clear
file = fopen('Datafile01_10.dat', 'r');
fscanf(file, ' Time (s) x (m) y (m)\n');
Table = fscanf(file, '%f %f %f\n', [3,11]);
fclose(file);
% Print on the screen
fprintf(' Time (s) x (m) y (m)\n');
fprintf('%10.1f %9.3f %9.3f\n', Table);
Command Window
>> Example01_10
K>> Table
Table =
0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000 0.7000 0.8000 0.9000 1.0000
0 0.3536 0.7071 1.0607 1.4142 1.7678 2.1213 2.4749 2.8284 3.1820 3.5355
0 0.3045 0.5109 0.6192 0.6294 0.5415 0.3555 0.0714 -0.3108 -0.7911 -1.3695
K>>
Paused in debugger


Editor - C:\Users\Lee\Documents\MATLAB\Chapter01\Example01_10.m
Example01_10.m
clear
v0 = 5; theta = pi/4; g = 9.81;
t = 0:0.1:1;
x = v0*cos(theta)*t;
y = v0*sin(theta)*t-g*t.^2/2;
Table = [t; x; y];
% Write to a file
file = fopen('Datafile01_10.dat', 'w+');
fprintf(file, ' Time (s)   x (m)   y (m)\n');
fprintf(file, '%10.1f %9.3f %9.3f\n', Table);
fclose(file);
% Read from the file
clear
file = fopen('Datafile01_10.dat', 'r');
fscanf(file, ' Time (s)   x (m)   y (m)\n');
Table = fscanf(file, '%f %f %f\n', [3,11]);
fclose(file);
% Print on the screen
fprintf(' Time (s)   x (m)   y (m)\n');
fprintf('%10.1f %9.3f %9.3f\n', Table);
Evaluate Selection F9
Open Selection Ctrl+D
Help on Selection F1
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Select All Ctrl+A
Wrap Comments Ctrl+J
Comment Ctrl+R
Uncomment Ctrl+T
Smart Indent Ctrl+I
Evaluate Current Section Ctrl+Enter
Insert Section Breaks Around Selection
Insert Text Markup
Function Browser Shift+F1
Function Hints Ctrl+F1
Code Folding
Split Screen

1.12 Binary File Input/Output

Example01_12.m: Binary File I/O

[2] Create a new script, type the following commands, save as Example01_12.m, and run the script. This program demonstrates input/output of a binary file.

```
clear
v0 = 5; theta = pi/4; g = 9.81;
t = 0:0.1:1;
x = v0*cos(theta)*t;
y = v0*sin(theta)*t-g*t.^2/2;
Table = [t; x; y];
% Write to a file
save('Datafile01_12');
% Read from the file
clear
load('Datafile01_12');
% Print on the screen
fprintf('  Time (s)     x (m)     y (m)\n');
fprintf('%10.1f %9.3f %9.3f\n', Table);
```

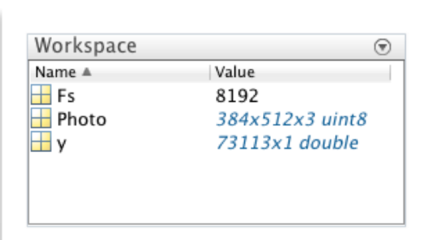
```
>> Example01_12
  Time (s)     x (m)     y (m)
       0.0     0.000     0.000
       0.1     0.354     0.305
       0.2     0.707     0.511
       0.3     1.061     0.619
       0.4     1.414     0.629
       0.5     1.768     0.542
       0.6     2.121     0.356
       0.7     2.475     0.071
       0.8     2.828    -0.311
       0.9     3.182    -0.791
       1.0     3.536    -1.369
>>
```

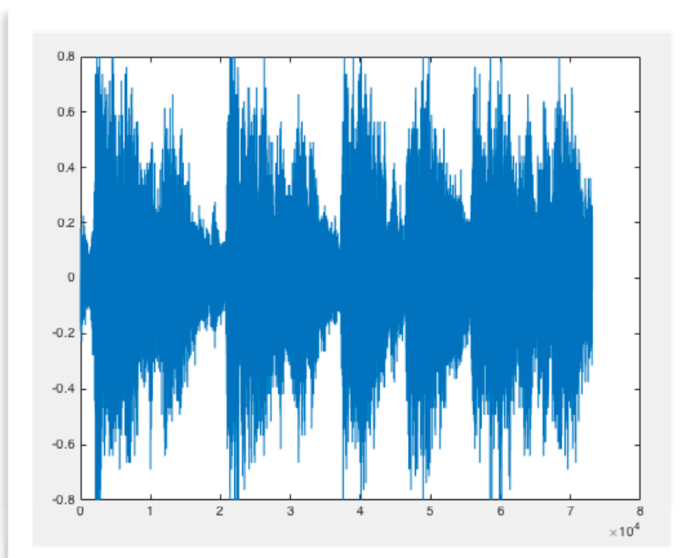
1.13 Images and Sounds

Example01_13.m: Images and Sounds

[2] This program displays an image [4], plays a song, and plots the audio signals of the song [5].

```
1   clear
2   Photo = imread('peppers.png');
3   image(photo)
4   axis image
5   load handel
6   sound(y, Fs)
7   figure
8   plot(y)
```





1.14 Flow Controls

Example01_14.m: For-Loops

[2] Create a new script, type the following commands, save as Example01_14.m, and run the script.

```
clear
v0 = 5; theta = pi/4; g = 9.81;
t = 0:0.1:1;
x = v0*cos(theta)*t;
y = v0*sin(theta)*t-g*t.^2/2;
fprintf('\n Time (s)  Distance (m)    Height (m)\n')
for k = 1:length(t)
    fprintf('%10.1f %9.3f %9.3f\n', t(k), x(k), y(k))
end
```

```
>> Example01_14
  Time (s)      x (m)      y (m)
       0.0      0.000      0.000
       0.1      0.354      0.305
       0.2      0.707      0.511
       0.3      1.061      0.619
       0.4      1.414      0.629
       0.5      1.768      0.542
       0.6      2.121      0.356
       0.7      2.475      0.071
       0.8      2.828     -0.311
       0.9      3.182     -0.791
       1.0      3.536     -1.369
>>
```

1.15 User-Defined Functions

Example01_15a.m: User-Defined Functions

[2] Create a program like this, save as Example01_15a.m, and run the program. This program demonstrates the creation and use of user-defined functions.

```
clear, global g
v0 = 5; theta = pi/4; g = 9.81;
t = 0:0.1:1;
[distance, height] = trajectory(v0, theta, t);
printTable(t, distance, height);

function [x, y] = trajectory(v0, angle, time)
global g
x = v0*cos(angle)*time;
y = v0*sin(angle)*time-g*time.^2/2;
end

function printTable(t, x, y)
fprintf('  Time (s)     x (m)     y (m)\n');
for k = 1:length(t)
    fprintf('%10.1f %9.3f %9.3f\n', t(k), x(k), y(k))
end
end
```

```
>> Example01_15a
  Time (s)      x (m)      y (m)
       0.0      0.000      0.000
       0.1      0.354      0.305
       0.2      0.707      0.511
       0.3      1.061      0.619
       0.4      1.414      0.629
       0.5      1.768      0.542
       0.6      2.121      0.356
       0.7      2.475      0.071
       0.8      2.828     -0.311
       0.9      3.182     -0.791
       1.0      3.536     -1.369
>>
```

Example01_15b.m: A Program with Input Arguments

[13] Create a program like this, save as Example01_15b.m, and run the program by typing (see [14])

```
>> Example01_15b(5, pi/4)
```

This program demonstrates the creation of a user-defined command with input arguments, here, `v0` and `theta`. Note that the dimmed statements are copied from Example01_15a.m.

```
function Example01_15b(v0, theta)
global g
g = 9.81;
t = 0:0.1:1;
[distance, height] = trajectory(v0, theta, t);
printTable(t, distance, height);
end

function [x, y] = trajectory(v0, angle, time)
global g
x = v0*cos(angle)*time;
y = v0*sin(angle)*time-g*time.^2/2;
end

function printTable(t, x, y)
fprintf('  Time (s)      x (m)      y (m)\n');
for k = 1:length(t)
    fprintf('%10.1f %9.3f %9.3f\n', t(k), x(k), y(k))
end
end
```

```
>> Example01_15b(5, pi/4)
  Time (s)      x (m)      y (m)
       0.0     0.000      0.000
       0.1     0.354      0.305
       0.2     0.707      0.511
       0.3     1.061      0.619
       0.4     1.414      0.629
       0.5     1.768      0.542
       0.6     2.121      0.356
       0.7     2.475      0.071
       0.8     2.828     -0.311
       0.9     3.182     -0.791
       1.0     3.536     -1.369
>>
```

1.16 Cell Arrays

```
>> Example01_16
Initial velocity = 5 m/s
Elevation angle = 45 degrees

  Time (s)      x (m)      y (m)
       0.0      0.000      0.000
       0.1      0.354      0.305
       0.2      0.707      0.511
       0.3      1.061      0.619
       0.4      1.414      0.629
       0.5      1.768      0.542
       0.6      2.121      0.356
       0.7      2.475      0.071
       0.8      2.828     -0.311
       0.9      3.182     -0.791
       1.0      3.536     -1.369
>>
```

Example01_16.m: Cell Arrays

[2] Create a program like this and run the program.
This program demonstrates the creation and use of **cell arrays**.

```
clear
v0 = 5; theta = pi/4; g = 9.81;
t = 0:0.1:1;
x = v0*cos(theta)*t;
y = v0*sin(theta)*t-g*t.^2/2;
Trajectory = {v0, theta, t, x, y};
printTrajectory(Trajectory)

function printTrajectory(Traj)
fprintf('Initial velocity = %.0f m/s\n', Traj{1})
fprintf('Elevation angle = %.0f degrees\n\n', Traj{2}*180/pi)
fprintf('  Time (s)     x (m)     y (m)\n');
for k = 1:length(Traj{3})
    fprintf('%10.1f %9.3f %9.3f\n', Traj{3}(k), Traj{4}(k), Traj{5}(k))
end
end
```

1.17 Structures

Example01_17.m: Structures

[2] Create a program like this and run the program.
This program demonstrates the creation and use of **structures**.

```
clear
v0 = 5; theta = pi/4; g = 9.81;
t = 0:0.1:1;
x = v0*cos(theta)*t;
y = v0*sin(theta)*t-g*t.^2/2;
Trajectory.velocity = v0;
Trajectory.angle = theta;
Trajectory.time = t;
Trajectory.distance = x;
Trajectory.height = y;
printTrajectory(Trajectory)

function printTrajectory(Traj)
fprintf('Initial velocity = %.0f m/s\n', Traj.velocity)
fprintf('Elevation angle = %.0f degrees\n\n', Traj.angle*180/pi)
fprintf('  Time (s)     x (m)     y (m)\n');
for k = 1:length(Traj.time)
    fprintf('%10.1f %9.3f %9.3f\n', ...
        Traj.time(k), Traj.distance(k), Traj.height(k))
end
end
```

```
>> Example01_17
Initial velocity = 5 m/s
Elevation angle = 45 degrees

  Time (s)     x (m)     y (m)
       0.0     0.000     0.000
       0.1     0.354     0.305
       0.2     0.707     0.511
       0.3     1.061     0.619
       0.4     1.414     0.629
       0.5     1.768     0.542
       0.6     2.121     0.356
       0.7     2.475     0.071
       0.8     2.828    -0.311
       0.9     3.182    -0.791
       1.0     3.536    -1.369
>>
```

1.18 Graphical User Interfaces (GUI)

Example01_18.m: GUI

[2] Create a program like this and run the program. This program creates a GUI as shown in [3-6], next page. →

```
clear
global g velocityBox angleBox
g = 9.81;
figure('Position', [30,70,500,400])
axes('Units', 'pixels', ...
    'Position', [50,80,250,250])
axis([0, 10, 0, 10])
xlabel('Distance (m)'), ylabel('Height (m)')
title('Trajectory of a Ball')

uicontrol('Style', 'text', ...
    'String', 'Initial velocity (m/s)', ...
    'Position', [330,300,150,20])
velocityBox = uicontrol('Style', 'edit', ...
    'String', '5', ...
    'Position', [363,280,80,20]);
uicontrol('Style', 'text', ...
    'String', 'Elevation angle (deg)', ...
    'Position', [330,240,150,20])
angleBox = uicontrol('Style', 'edit', ...
    'String', '45', ...
    'Position', [363,220,80,20]);
uicontrol('Style', 'pushbutton', ...
    'String', 'Throw', ...
    'Position', [363,150,80,30], ...
    'Callback', @pushbuttonCallback)

function pushbuttonCallback(pushButton, ~)
global g velocityBox angleBox
v0 = str2double(velocityBox.String);
theta = str2double(angleBox.String)*pi/180;
t1 = 2*v0*sin(theta)/g;
t = 0:0.01:t1;
x = v0*cos(theta)*t;
y = v0*sin(theta)*t-g*t.^2/2;
hold on
comet(x, y)
end
```

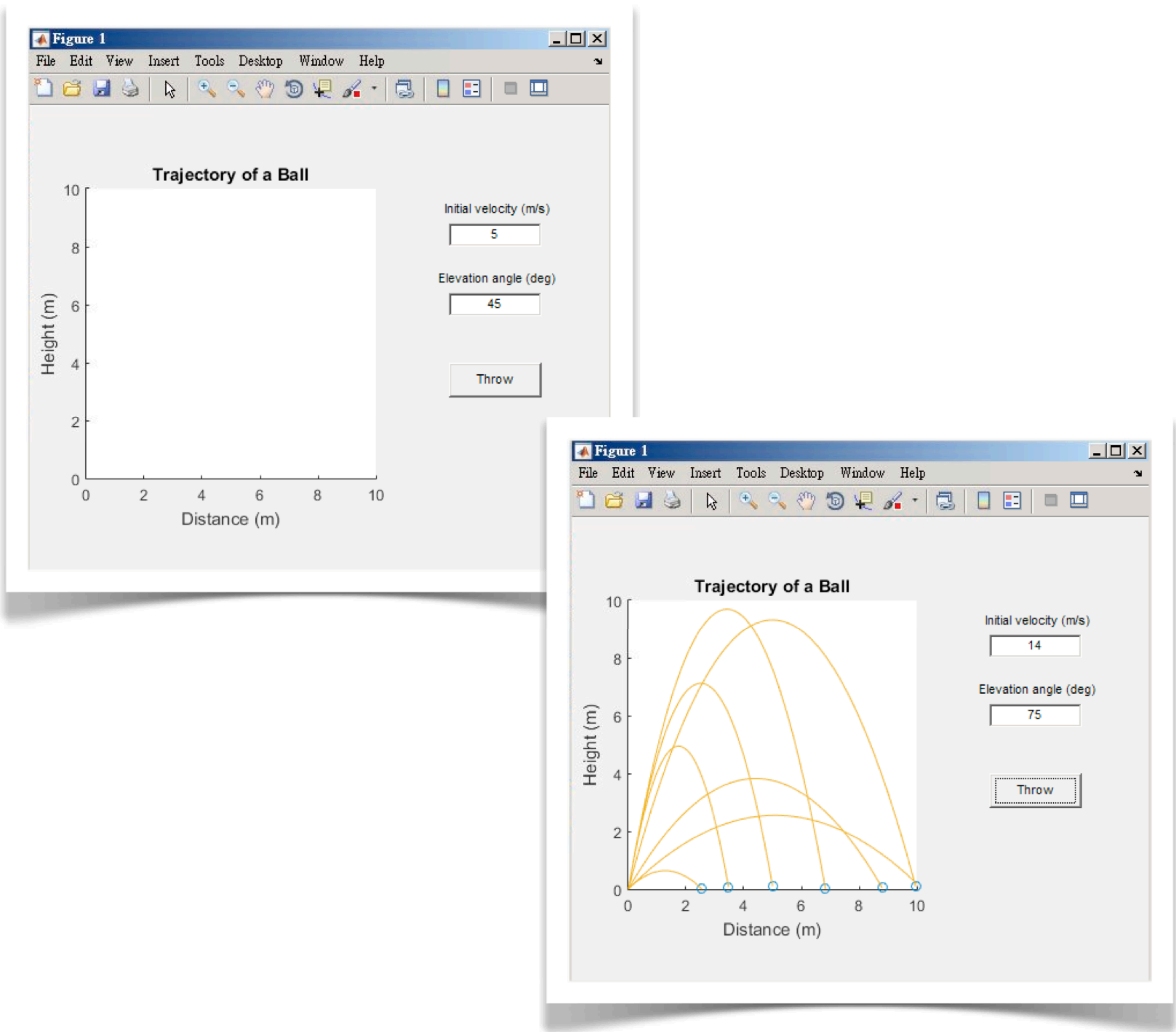
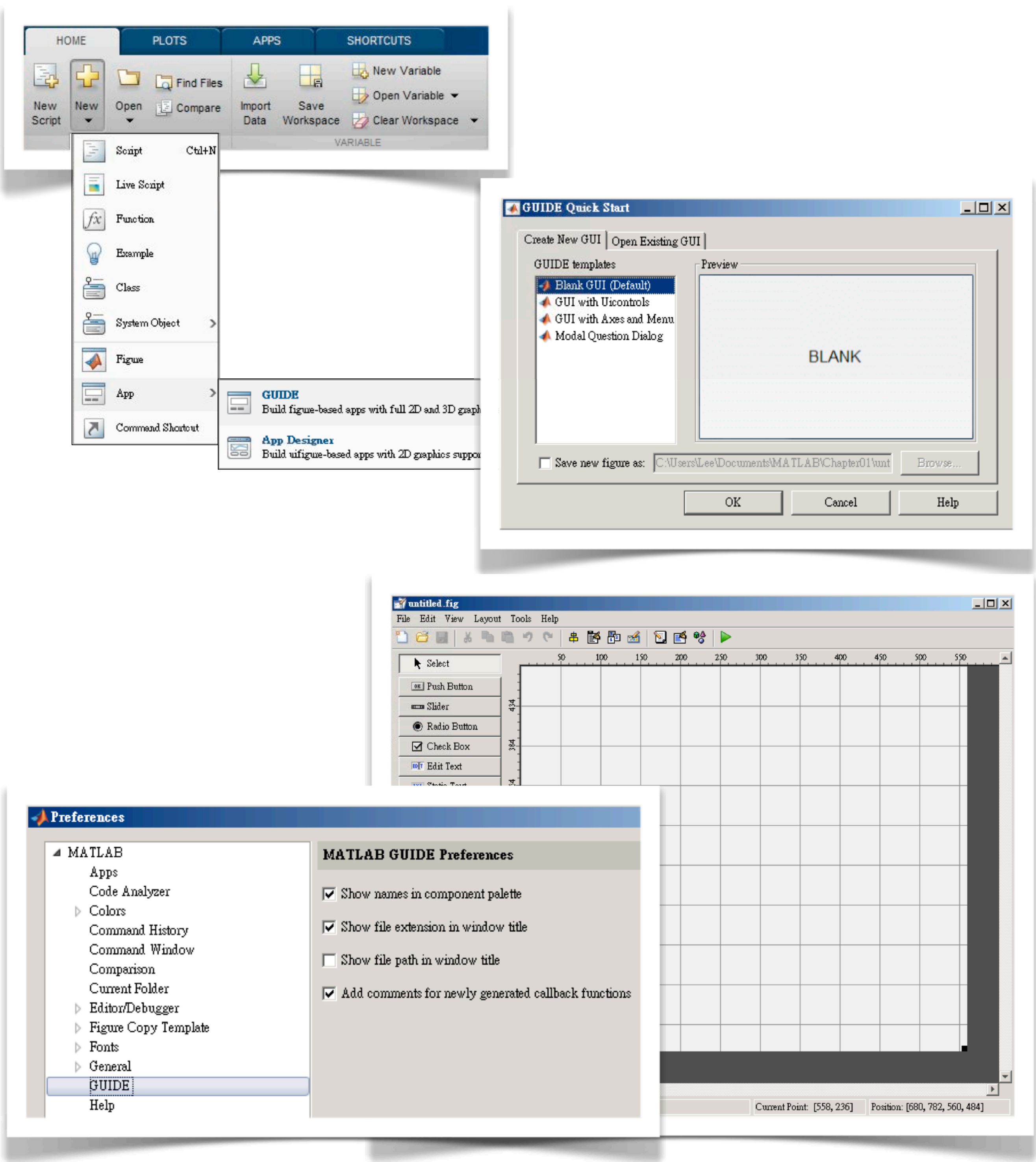
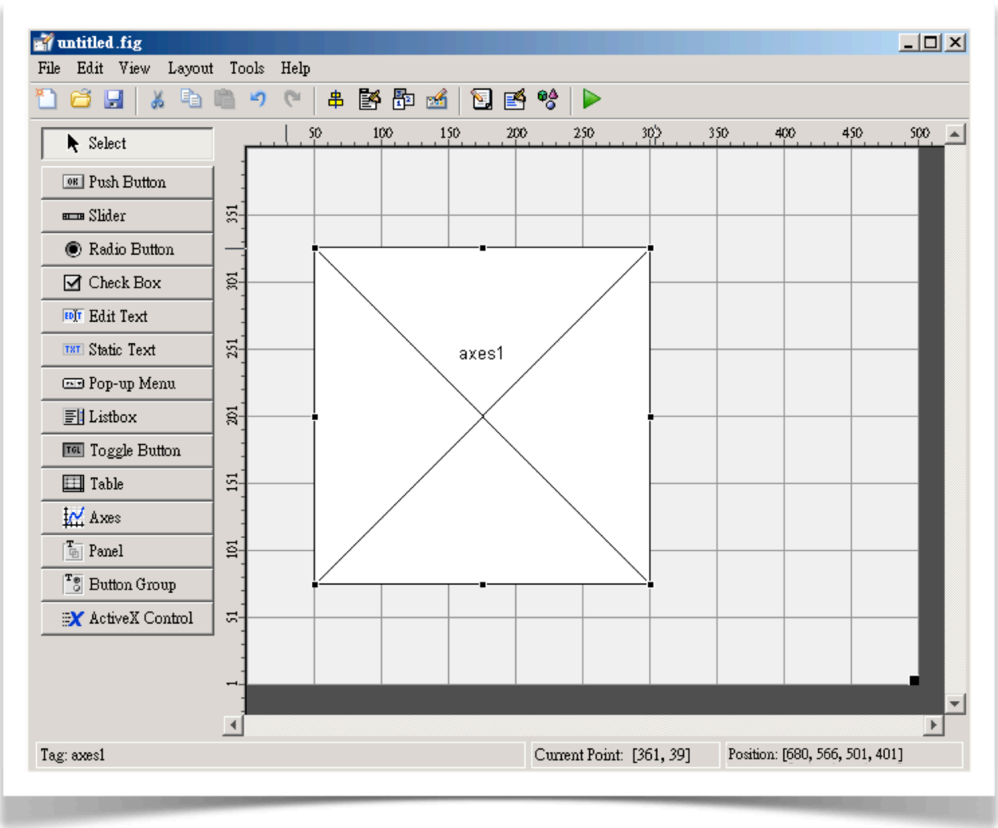
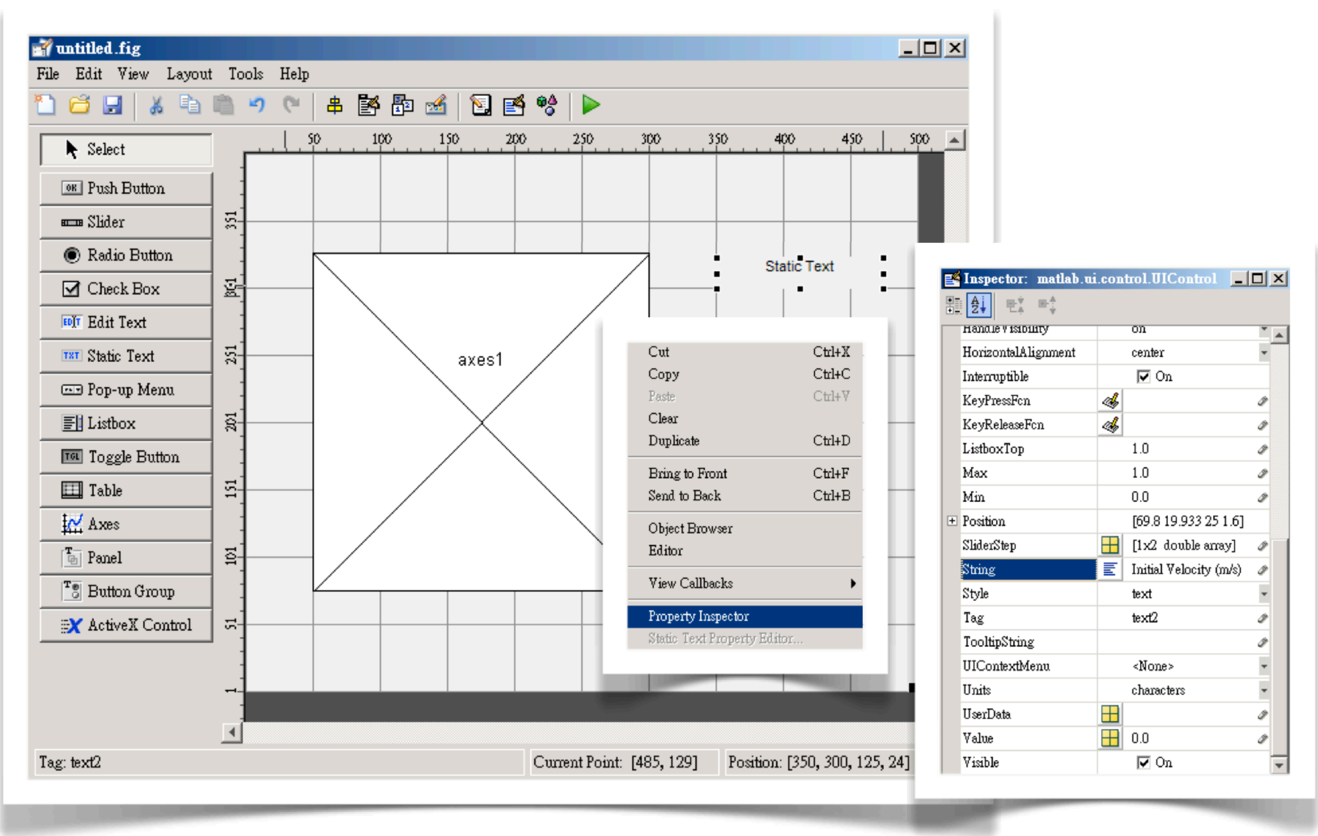


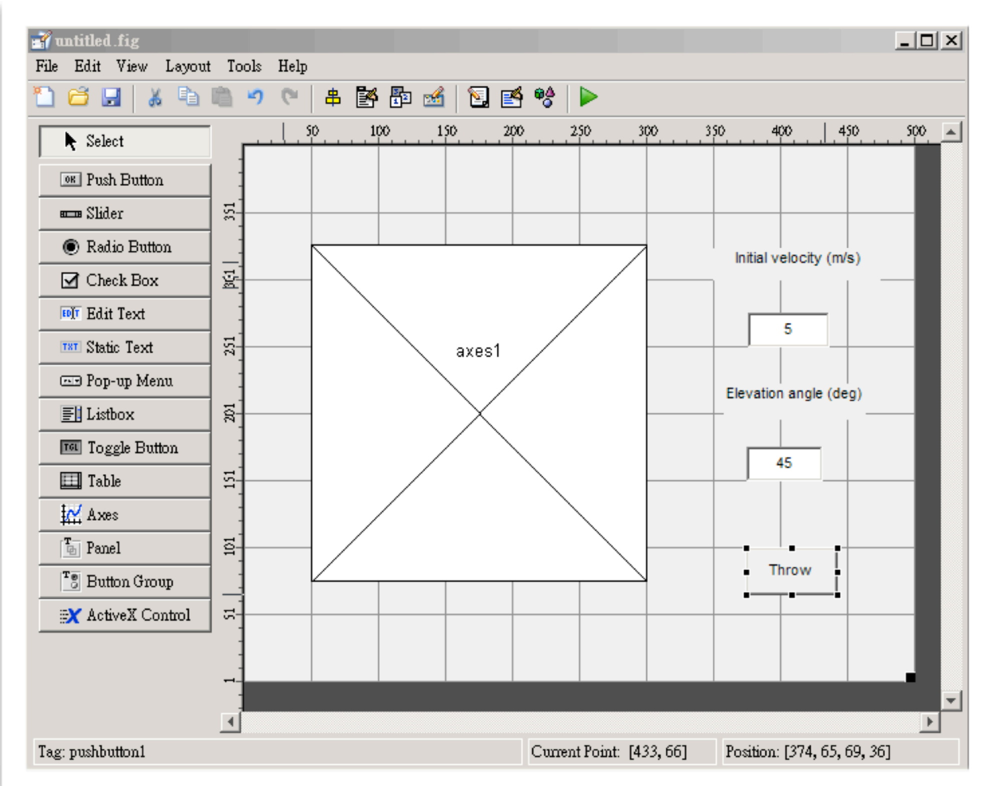
Figure 1
File Edit View Insert Tools Desktop Window Help
Trajectory of a Ball
Height (m)
Distance (m)
Initial velocity (m/s)
5
Elevation angle (deg)
45
Throw
Figure 1
File Edit View Insert Tools Desktop Window Help
Trajectory of a Ball
Height (m)
Distance (m)
Initial velocity (m/s)
14
Elevation angle (deg)
75
Throw

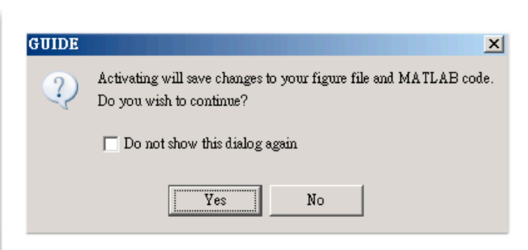
1.19 GUIDE

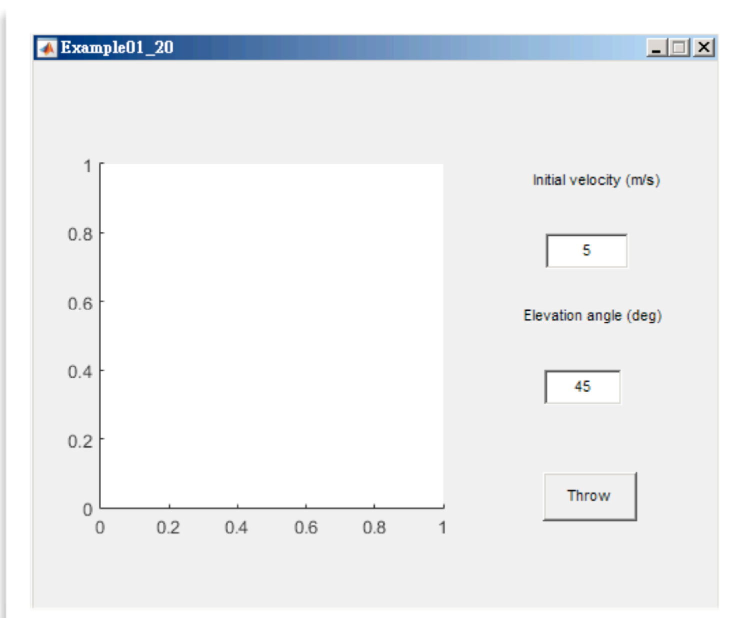

HOME
PLOTS
APPS
SHORTCUTS
New Script
New
Open
Find Files
Compare
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
VARIABLE
Script Ctrl+N
Live Script
Function
Example
Class
System Object
Figure
App
Command Shortcut
GUIDE
Build figure-based apps with full 2D and 3D graph
App Designer
Build uifigure-based apps with 2D graphics suppo
GUIDE Quick Start
Create New GUI
Open Existing GUI
GUIDE templates
Blank GUI (Default)
GUI with Uicontrols
GUI with Axes and Menu
Modal Question Dialog
Preview
BLANK
Save new figure as:
C:\Users\Lee\Documents\MATLAB\Chapter01\unt
Browse...
OK
Cancel
Help
untitled.fig
File Edit View Layout Tools Help
Select
Push Button
Slider
Radio Button
Check Box
Edit Text
Current Point: [558, 236]
Position: [680, 782, 560, 484]
Preferences
MATLAB
Apps
Code Analyzer
Colors
Command History
Command Window
Comparison
Current Folder
Editor/Debugger
Figure Copy Template
Fonts
General
GUIDE
Help
MATLAB GUIDE Preferences
Show names in component palette
Show file extension in window title
Show file path in window title
Add comments for newly generated callback functions


untitled.fig
File Edit View Layout Tools Help
Select
Push Button
Slider
Radio Button
Check Box
Edit Text
Static Text
Pop-up Menu
Listbox
Toggle Button
Table
Axes
Panel
Button Group
ActiveX Control
axes1
Tag: axes1
Current Point: [361, 39]
Position: [680, 566, 501, 401]


untitled.fig
File Edit View Layout Tools Help
Select
Push Button
Slider
Radio Button
Check Box
Edit Text
Static Text
Pop-up Menu
Listbox
Toggle Button
Table
Axes
Panel
Button Group
ActiveX Control
axes1
Static Text
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Clear
Duplicate Ctrl+D
Bring to Front Ctrl+F
Send to Back Ctrl+B
Object Browser
Editor
View Callbacks
Property Inspector
Static Text Property Editor...
Tag: text2
Current Point: [485, 129]
Position: [350, 300, 125, 24]
Inspector: matlab.ui.control.UIControl
HorizontalAlignment center
Interruptible On
KeyPressFcn
KeyReleaseFcn
ListboxTop 1.0
Max 1.0
Min 0.0
Position [69.8 19.933 25 1.6]
SliderStep [1x2 double array]
String Initial Velocity (m/s)
Style text
Tag text2
TooltipString
UIContextMenu <None>
Units characters
UserData
Value 0.0
Visible On


untitled.fig
File Edit View Layout Tools Help
Select
Push Button
Slider
Radio Button
Check Box
Edit Text
Static Text
Pop-up Menu
Listbox
Toggle Button
Table
Axes
Panel
Button Group
ActiveX Control
axes1
Initial velocity (m/s)
5
Elevation angle (deg)
45
Throw
Tag: pushbutton1
Current Point: [433, 66]
Position: [374, 65, 69, 36]


GUIDE
Activating will save changes to your figure file and MATLAB code.
Do you wish to continue?
Do not show this dialog again
Yes
No

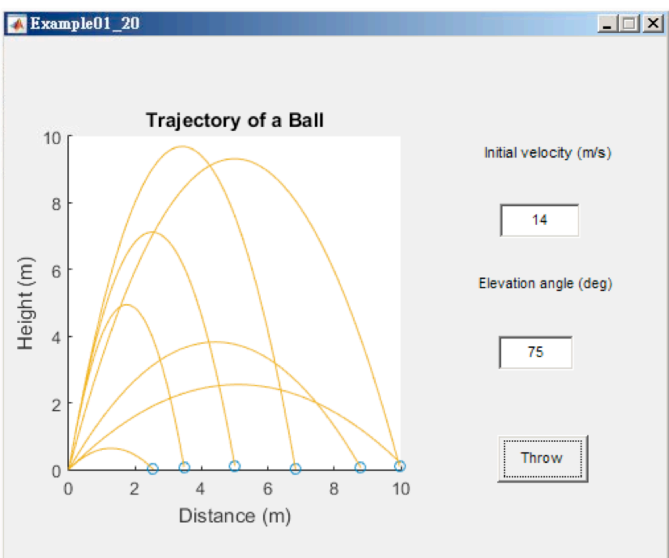

Example01_20
Initial velocity (m/s)
5
Elevation angle (deg)
45
Throw

```
% --- Executes just before Example01_19 is made visible.
function Example01_19_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Example01_19 (see VARARGIN)

% Choose default command line output for Example01_19
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Example01_19 wait for user response (see UIRESUME)
% uiwait(handles.figure1);
axis([0, 10, 0, 10])
xlabel('Distance (m)'), ylabel('Height (m)')
title('Trajectory of a Ball')
```

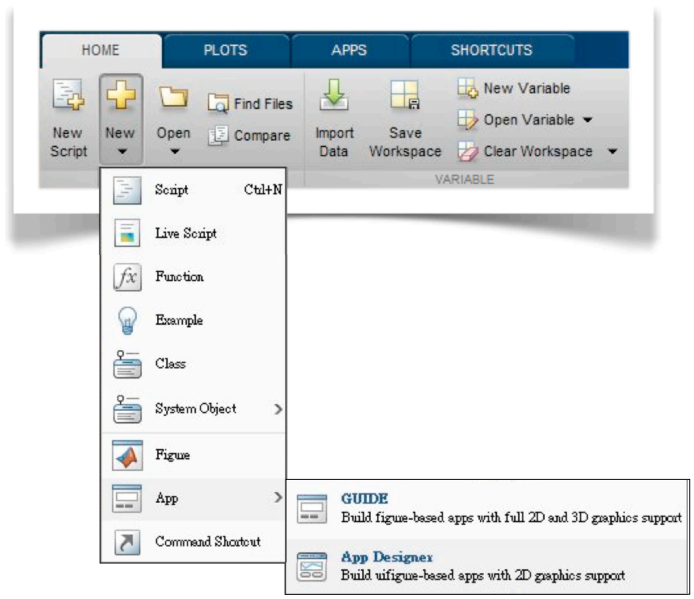
```
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
global velocityBox
velocityBox = hObject;
```

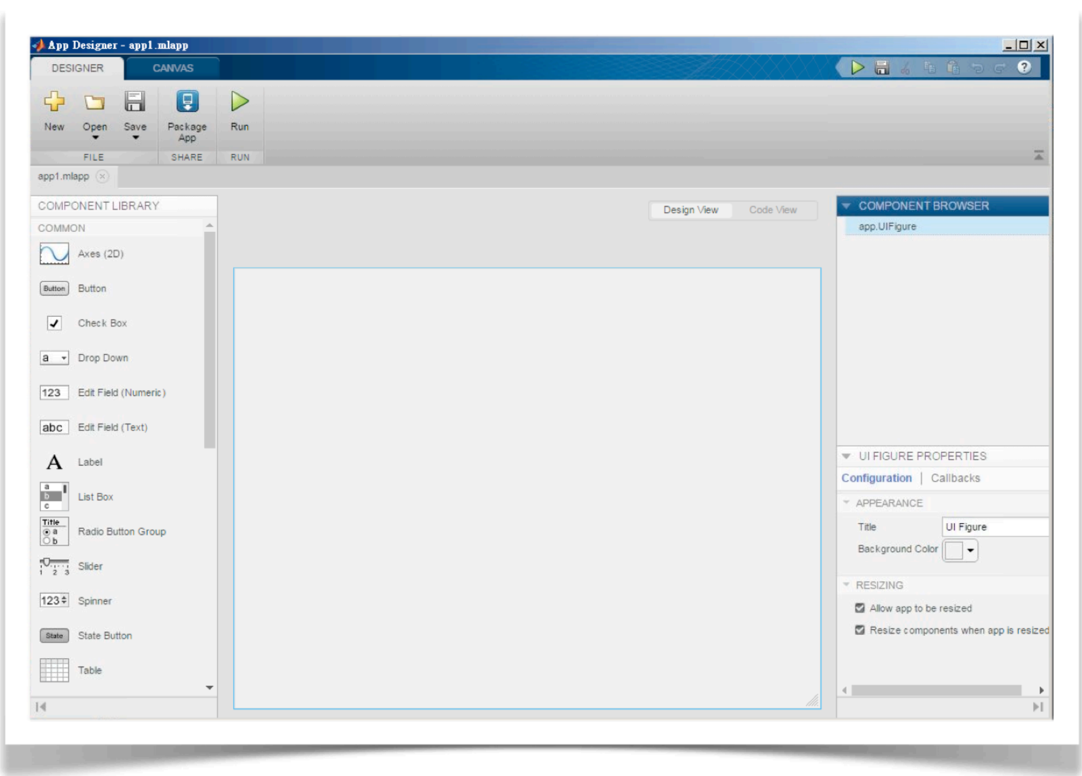
```
% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
global angleBox
angleBox = hObject;
```

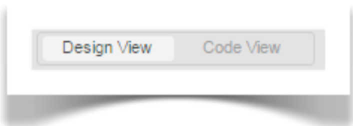
```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global velocityBox angleBox
g = 9.81;
v0 = str2double(velocityBox.String);
theta = str2double(angleBox.String)*pi/180;
t1 = 2*v0*sin(theta)/g;
t = 0:0.01:t1;
x = v0*cos(theta)*t;
y = v0*sin(theta)*t-g*t.^2/2;
hold on
comet(x, y)
```

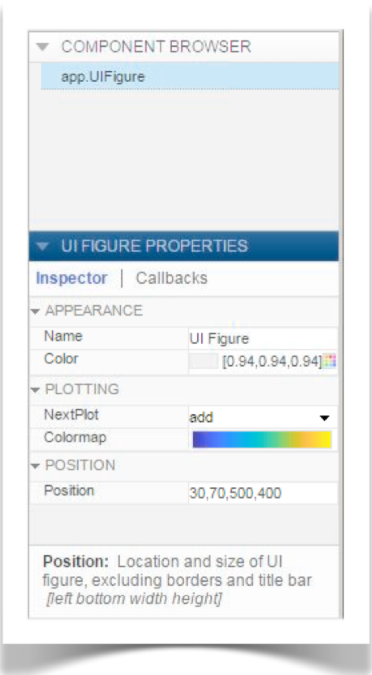


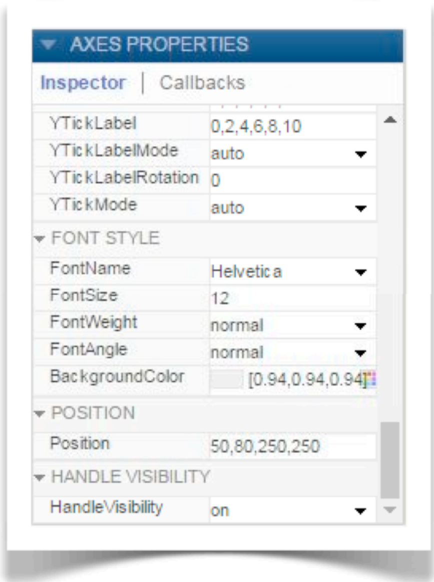
1.20 App Designer

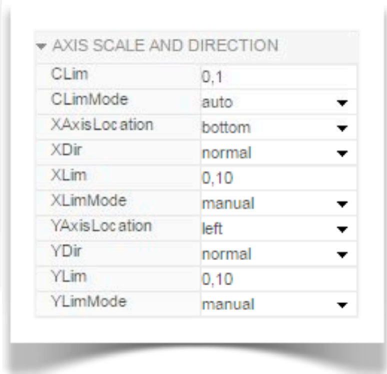


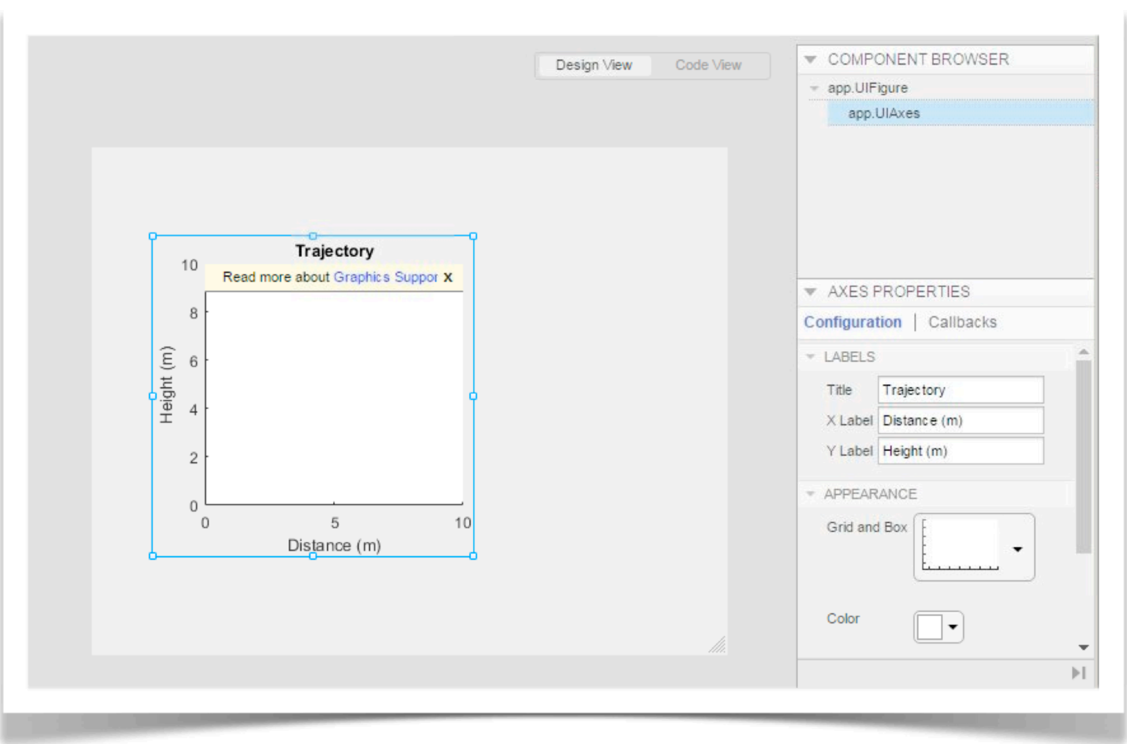


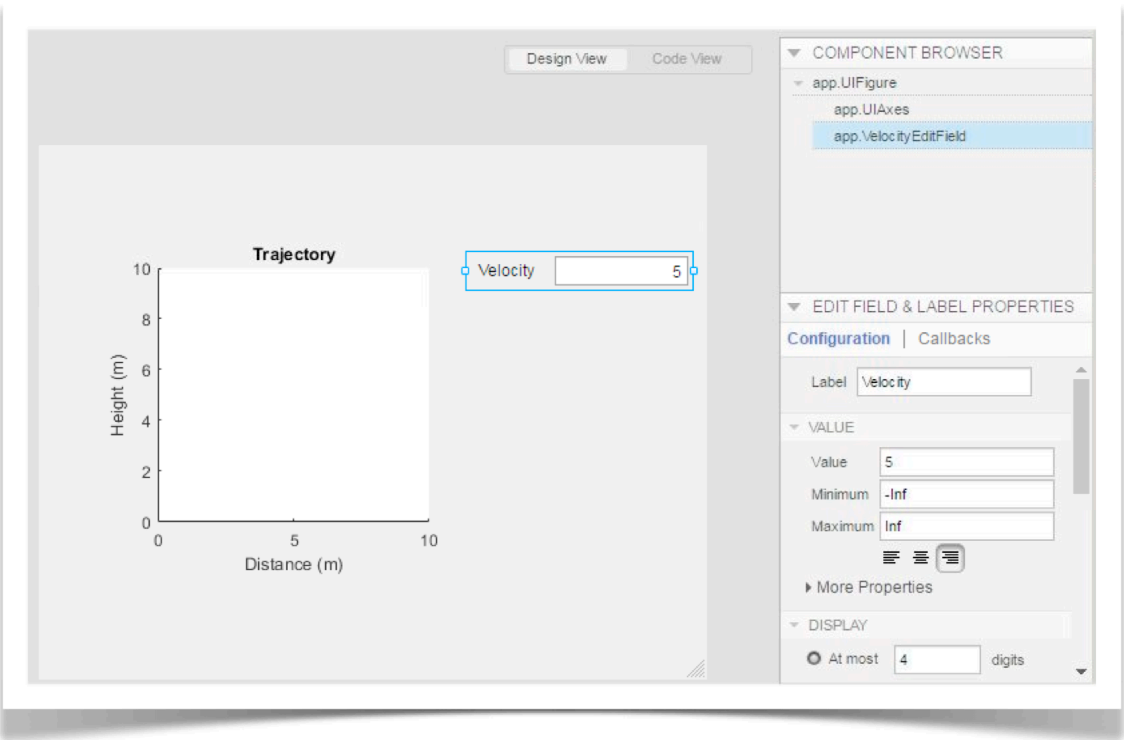

Design View
Code View

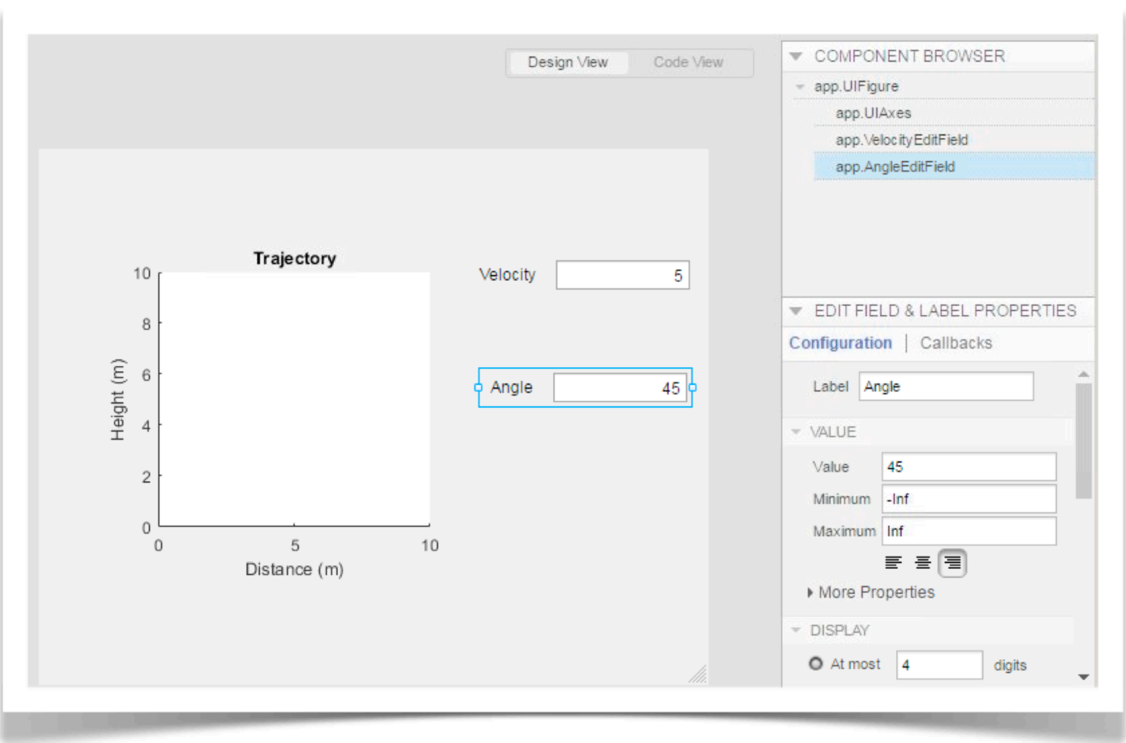

COMPONENT BROWSER
app.UIFigure
UI FIGURE PROPERTIES
Inspector | Callbacks
APPEARANCE
Name UI Figure
Color [0.94,0.94,0.94]
PLOTTING
NextPlot add
Colormap
POSITION
Position 30,70,500,400
Position: Location and size of UI figure, excluding borders and title bar
[left bottom width height]

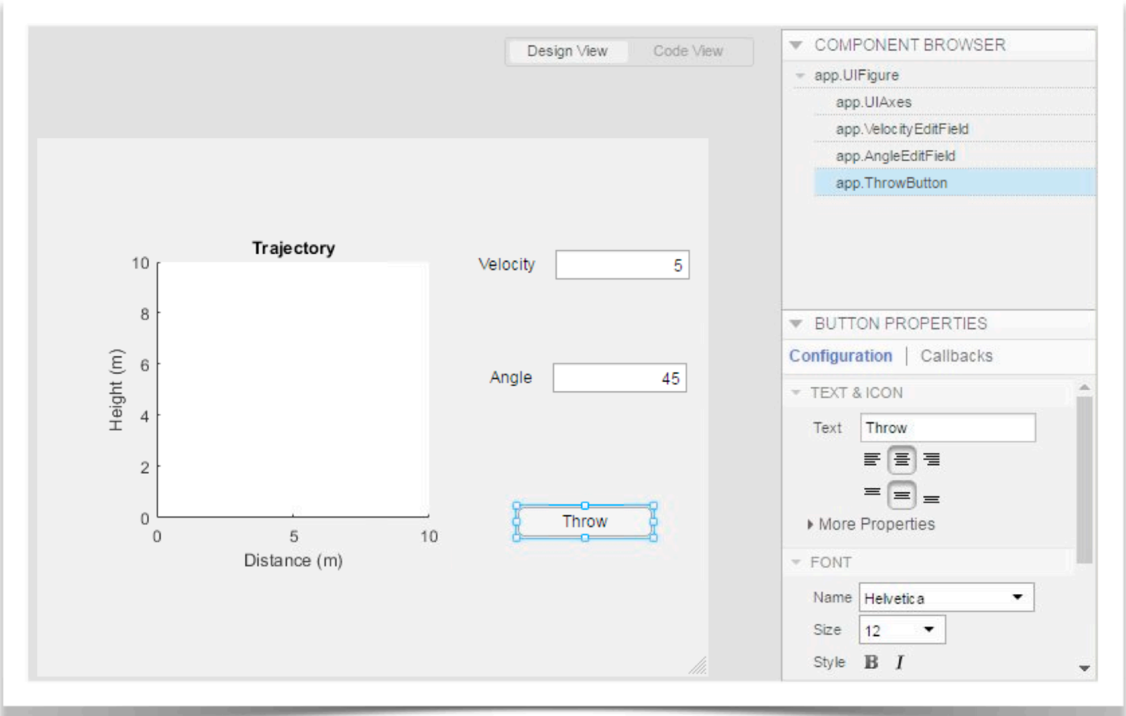

AXES PROPERTIES
Inspector | Callbacks
YTickLabel 0,2,4,6,8,10
YTickLabelMode auto
YTickLabelRotation 0
YTickMode auto
FONT STYLE
FontName Helvetica
FontSize 12
FontWeight normal
FontAngle normal
BackgroundColor [0.94,0.94,0.94]
POSITION
Position 50,80,250,250
HANDLE VISIBILITY
HandleVisibility on

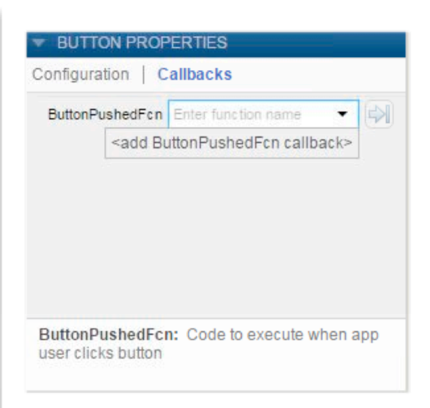

AXIS SCALE AND DIRECTION
CLim 0,1
CLimMode auto
XAxisLocation bottom
XDir normal
XLim 0,10
XLimMode manual
YAxisLocation left
YDir normal
YLim 0,10
YLimMode manual


Design View
Code View
Trajectory
Read more about Graphics Suppor x
Height (m)
Distance (m)
COMPONENT BROWSER
app.UIFigure
app.UIAxes
AXES PROPERTIES
Configuration | Callbacks
LABELS
Title Trajectory
X Label Distance (m)
Y Label Height (m)
APPEARANCE
Grid and Box
Color


Design View
Code View
Trajectory
Height (m)
Distance (m)
Velocity
5
COMPONENT BROWSER
app.UIFigure
app.UIAxes
app.VelocityEditField
EDIT FIELD & LABEL PROPERTIES
Configuration | Callbacks
Label Velocity
VALUE
Value 5
Minimum -Inf
Maximum Inf
More Properties
DISPLAY
At most 4 digits


Design View
Code View
Trajectory
Height (m)
Distance (m)
Velocity
5
Angle
45
COMPONENT BROWSER
app.UIFigure
app.UIAxes
app.VelocityEditField
app.AngleEditField
EDIT FIELD & LABEL PROPERTIES
Configuration | Callbacks
Label Angle
VALUE
Value 45
Minimum -Inf
Maximum Inf
More Properties
DISPLAY
At most 4 digits

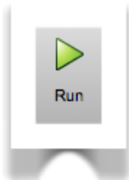


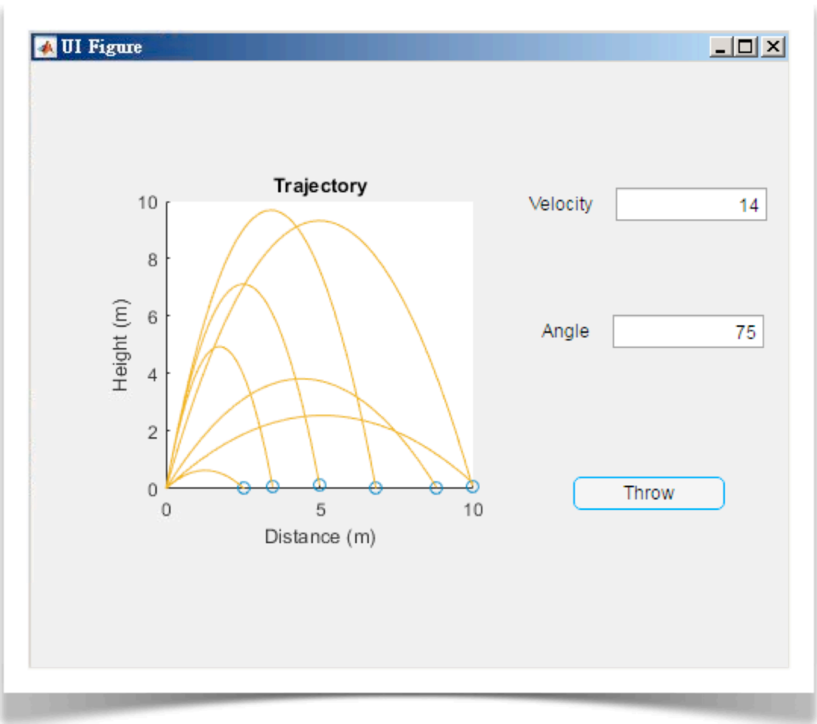


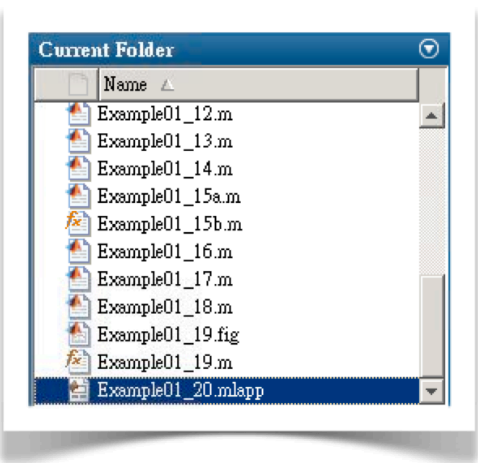

```
% Button pushed function: ThrowButton
function ThrowButtonPushed(app, event)

end
```

```
% Button pushed function: ThrowButton
function ThrowButtonPushed(app, event)
    g = 9.81;
    v0 = app.VelocityEditField.Value;
    theta = app.AngleEditField.Value*pi/180;
    t1 = 2*v0*sin(theta)/g;
    t = 0:0.01:t1;
    x = v0*cos(theta)*t;
    y = v0*sin(theta)*t-g*t.^2/2;
    hold(app.UIAxes, 'on')
    comet(app.UIAxes, x, y)
end
```


Save


Run


UI Figure
Trajectory
Height (m)
Distance (m)
0
2
4
6
8
10
0
5
10
Velocity
14
Angle
75
Throw

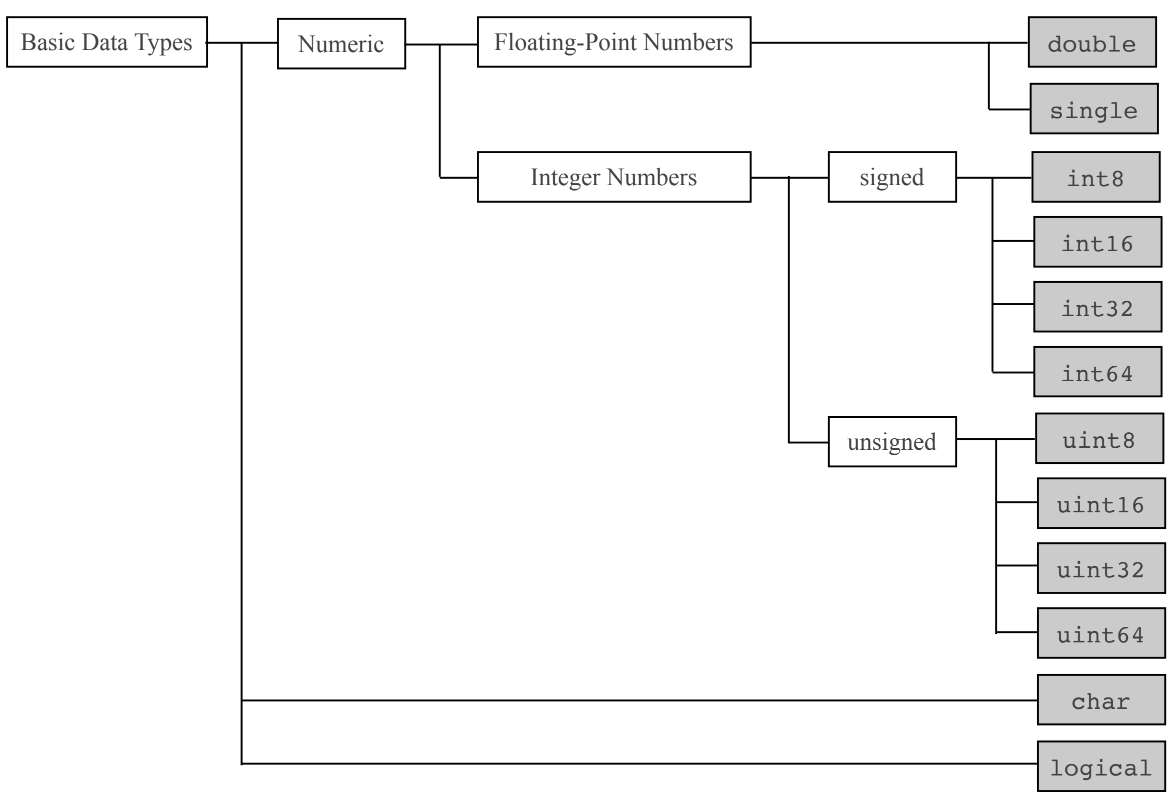

Current Folder
Name
Example01_12.m
Example01_13.m
Example01_14.m
Example01_15a.m
Example01_15b.m
Example01_16.m
Example01_17.m
Example01_18.m
Example01_19.fig
Example01_19.m
Example01_20.mlapp

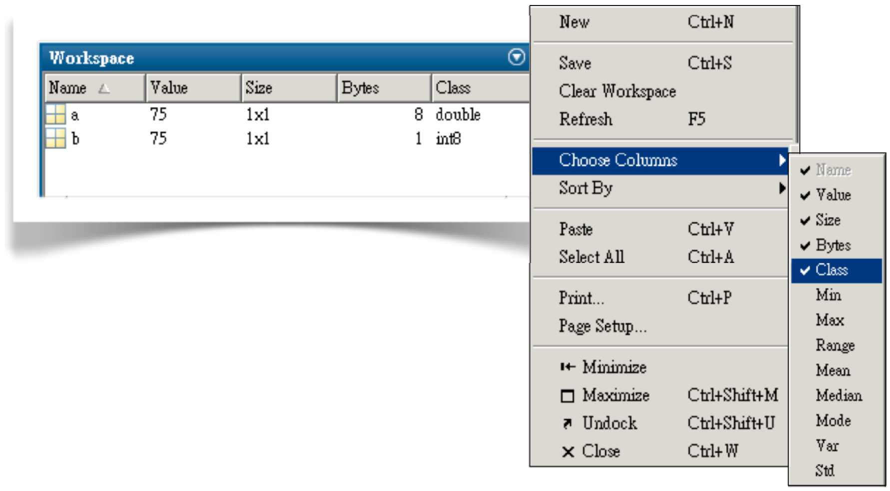
Chapter 2

Data Types, Operators, and Expressions

An expression is a syntactic combination of **numbers**, **variables**, **operators**, and **functions**. An expression always results in a **value**. The right-hand side of an assignment statement is always an expression. You may notice that most of the statements we demonstrated in Chapter 1 are assignment statements. It is fair to say that expressions are the most important building block of a program.

2.1 Unsigned Integers




Workspace
Name
Value
Size
Bytes
Class
a
75
1x1
8
double
b
75
1x1
1
int8
New Ctrl+N
Save Ctrl+S
Clear Workspace
Refresh F5
Choose Columns
Sort By
Paste Ctrl+V
Select All Ctrl+A
Print... Ctrl+P
Page Setup...
Minimize
Maximize Ctrl+Shift+M
Undock Ctrl+Shift+U
Close Ctrl+W
Name
Value
Size
Bytes
Class
Min
Max
Range
Mean
Median
Mode
Var
Std

Example02_01.m: Unsigned Integers

[5] These statements demonstrates the concepts given in the last page. A **Command Window** session is shown in [6].

```
clear
d = 75
u = uint8(d)
bits = bitget(u, 1:8)
bits = fliplr(bits)
```

```
>> clear
>> d = 75
d =
    75
>> u = uint8(d)
u =
  uint8
   75
>> bits = bitget(u, 1:8)
bits =
  1×8 uint8 row vector
   1    1    0    1    0    0    1    0
>> bits = fliplr(bits)
bits =
  1×8 uint8 row vector
   0    1    0    0    1    0    1    1
>>
```

Table 2.1 Unsigned Integer Numbers				
Conversion Function	Function to find the minimum value	Minimum value	Function to find the maximum value	Maximum value
`uint8`	`intmin('uint8')`	0	`intmax('uint8')`	255
`uint16`	`intmin('uint16')`	0	`intmax('uint16')`	65535
`uint32`	`intmin('uint32')`	0	`intmax('uint32')`	4294967295
`uint64`	`intmin('uint64')`	0	`intmax('uint64')`	18446744073709551615
Details and More: Help>MATLAB>Language Fundamentals>Data Types>Numeric Types				

2.2 Signed Integers

Table 2.2a Unsigned/Signed Representation

Bit pattern	Unsigned value	Signed value
`000`	0	0
`001`	1	1
`010`	2	2
`011`	3	3
`100`	4	-4
`101`	5	-3
`110`	6	-2
`111`	7	-1
`00000000`	0	0
`11111111`	255	-1
`01111111`	127	127
`10000000`	128	-128

Details and More: Wikipedia>Two's complement

Table 2.2b Signed Integer Numbers

Conversion Function	Function to find the minimum value	Minimum value	Function to find the maximum value	Maximum value
`int8`	`intmin('int8')`	-128	`intmax('int8')`	127
`int16`	`intmin('int16')`	-32768	`intmax('int16')`	32767
`int32`	`intmin('int32')`	-2147483648	`intmax('int32')`	2147483647
`int64`	`intmin('int64')`	-9223372036854775808	`intmax('int64')`	9223372036854775807

Details and More: Help>MATLAB>Language Fundamentals>Data Types>Numeric Types

Example02_02.m: Signed Integers

[3] These statements demonstrates some concepts about signed integers. A **Command Window** session is shown in [4].

```
clear
d = 200
u = uint8(d)
bits = fliplr(bitget(u, 1:8))
t = int8(u)
a = int16(u)
s = typecast(u, 'int8')
bits = fliplr(bitget(s, 1:8))
```

```
>> clear
>> d = 200
d =
   200
>> u = uint8(d)
u =
  uint8
   200
>> bits = fliplr(bitget(u, 1:8))
bits =
  1×8 uint8 row vector
   1    1    0    0    1    0    0    0
>> t = int8(u)
t =
  int8
   127
>> a = int16(u)
a =
  int16
   200
>> s = typecast(u, 'int8')
s =
  int8
   -56
>> bits = fliplr(bitget(s, 1:8))
bits =
  1×8 int8 row vector
   1    1    0    0    1    0    0    0
>>
```

2.3 Floating-Point Numbers

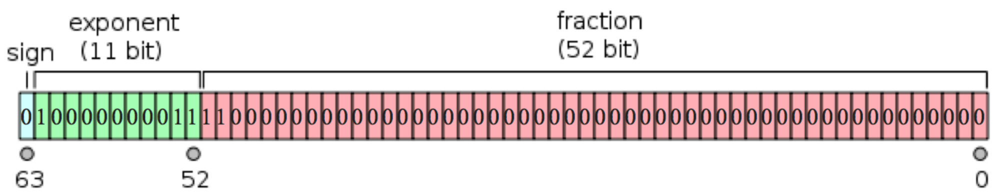


Table 2.3a Floating-Point Numbers

Conversion Function	Function to find the minimum value	Minimum value	Function to find the maximum value	Maximum value
`double`	`realmin('double')`	2.2251e-308	`realmax('double')`	1.7977e+308
`single`	`realmin('single')`	1.1755e-38	`realmax('single')`	3.4028e+38

Details and More: Help>MATLAB>Language Fundamentals>Data Types>Numeric Types

Example02_03a.m: Floating-Point Numbers

[3] These statements confirm that the decimal number 28 is indeed represented by the bit pattern in [1], last page. A **Command Window** session is shown in [4].

```
1   clear
2   d = 28
3   a = typecast(d, 'uint64')
4   b = dec2bin(a, 64)
```

Line 2 creates a double-precision floating-point number 28 and stores it in the variable `d`. The bit pattern supposedly is the one in [1], last page.

In line 3, the function `typecast` preserves the 64-bit pattern while change its type to `uint64`. Now, the bit pattern is interpreted as a value of 4628574517030027264 (see line 12), which can be calculated by

$$2^{62} + 2^{53} + 2^{52} + 2^{51} + 2^{50} = 4628574517030027264$$

Line 4 demonstrates another way (than using `bitget` and `fliplr`) to display the bit pattern. The function `dec2bin(a,64)` retrieves the bit pattern from an integer number `a` and outputs the bit pattern in a text form (i.e., a string, to be introduced in the next section). The result is shown in line 15, the same as the one in [1], last page.

```
5   >> clear
6   >> d = 28
7   d =
8       28
9   >> a = typecast(d, 'uint64')
10  a =
11    uint64
12    4628574517030027264
13  >> b = dec2bin(a, 64)
14  b =
15      '0100000000111100000000000000000000000000000000000000000000000000'
16  >>
```

Example02_03b.m: Precision of Floating-Point Numbers

[5] These statements introduce some concepts about the precision of floating-point numbers. A **Command Window** session is shown in [6], next page. →

```
clear
format short
format compact
a = 1234.56789012345678901234
fprintf('%.20f\n', a)
format long
a
single(a)
```

```
>> clear
>> format short
>> format compact
>> a = 1234.5678901234567890l234
a =
   1.2346e+03
>> fprintf('%.20f\n', a)
1234.56789012345689116046
>> format long
>> a
a =
       1.234567890123457e+03
>> single(a)
ans =
  single
   1.2345679e+03
>>
```

Table 2.3b Numeric Output Format

Function	Description or Example
`format compact`	Suppress blank lines
`format loose`	Add blank lines
`format short`	3.1416
`format long`	3.141592653589793
`format shortE`	3.1416e+00
`format longE`	3.141592653589793e+00
`format shortG`	`short` or `shortE`
`format longG`	`long` or `longE`
`format shortEng`	Exponent is a multiple of 3
`format longEng`	Exponent is a multiple of 3
`format +`	Display the sign (+/-)
`format bank`	Currency format; 3.14
`format hex`	400921fb54442d18
`format rat`	Rational; 355/133

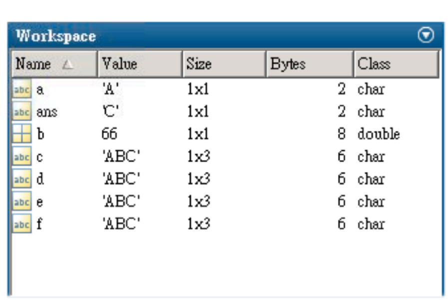
Details and More: >> doc format

2.4 Characters and Strings

Example02_04a.m: Characters

[2] These statements demonstrates some concepts about **characters** and **strings**. A **Command Window** session is shown in [3] and the **Workspace** is shown in [4-5].

```
clear
a = 'A'
b = a + 1
char(65)
char('A' + 2)
c = ['A', 'B', 'C']
d = ['AB', 'C']
e = ['A', 66, 67]
f = 'ABC'
f(1)
f(2)
f(3)
```



Name	Value	Size	Bytes	Class
a	'A'	1x1	2	char
ans	'C'	1x1	2	char
b	66	1x1	8	double
c	'ABC'	1x3	6	char
d	'ABC'	1x3	6	char
e	'ABC'	1x3	6	char
f	'ABC'	1x3	6	char

```
>> clear
>> a = 'A'
a =
     'A'
>> b = a + 1
b =
     66
>> char(65)
ans =
     'A'
>> char('A' + 2)
ans =
     'C'
>> c = ['A', 'B', 'C']
c =
     'ABC'
>> d = ['AB', 'C']
d =
     'ABC'
>> e = ['A', 66, 67]
e =
     'ABC'
>> f = 'ABC'
f =
     'ABC'
>> f(1)
ans =
     'A'
>> f(2)
ans =
     'B'
>> f(3)
ans =
     'C'
```

Example02_04b.m: ASCII Codes

[7] MATLAB stores characters according to ASCII Code. ASCII codes 32-126 represent all printable characters on a standard keyboard. This example prints a table of characters corresponding to the ASCII Codes 32-126 (see the output in [8], next page). →

```
clear
fprintf('    0 1 2 3 4 5 6 7 8 9\n')
for row = 3:12
    fprintf('%2d  ', row)
    for column = 0:9
        code = row*10+column;
        if (code < 32) || (code > 126)
           fprintf('  ')
        else
           fprintf('%c ', code)
        end
    end
    fprintf('\n')
end
```

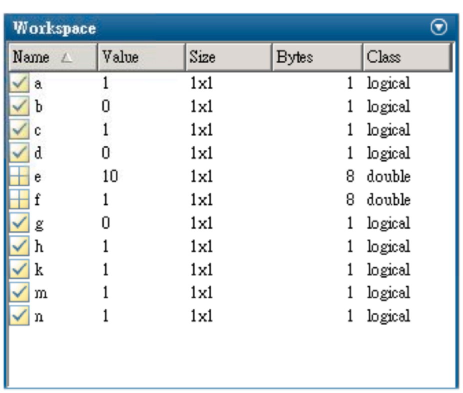
	0	1	2	3	4	5	6	7	8	9
3				!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~			

2.5 Logical Data

Example02_05.m: Logical Data Type

[2] These statements demonstrates some concepts about **logical** data. A **Command Window** session and the **Workspace** is shown in [3, 4], respectively.

```
clear
a = true
b = false
c = 6 > 5
d = 6 < 5
e = (6 > 5)*10
f = false*10+true*2
g = (6 > 5) & (6 < 5)
h = (6 > 5) | (6 < 5)
k = logical(5)
m = 5 | 0
n = (-2) & 'A'
```



Name	Value	Size	Bytes	Class
a	1	1x1	1	logical
b	0	1x1	1	logical
c	1	1x1	1	logical
d	0	1x1	1	logical
e	10	1x1	8	double
f	1	1x1	8	double
g	0	1x1	1	logical
h	1	1x1	1	logical
k	1	1x1	1	logical
m	1	1x1	1	logical
n	1	1x1	1	logical

```
>> clear
>> a = true
a =
  logical
   1
>> b = false
b =
  logical
   0
>> c = 6 > 5
c =
  logical
   1
>> d = 6 < 5
d =
  logical
   0
>> e = (6 > 5)*10
e =
    10
>> f = false*10+true*2
f =
     2
>> g = (6 > 5) & (6 < 5)
g =
  logical
   0
>> h = (6 > 5) | (6 < 5)
h =
  logical
   1
>> k = logical(5)
k =
  logical
   1
>> m = 5 | 0
m =
  logical
   1
>> n = (-2) & 'A'
n =
  logical
   1
```

Table 2.5a Rules of Logical `and` (`&`)

AND (&)	true	false
true	true	false
false	false	false

Table 2.5b Rules of Logical `or` (`|`)

OR (|)	true	false
true	true	true
false	true	false

2.6 Arrays

Example02_06a.m

[2] Type the following commands (also see [3]).

```
clear
a = 5
b = [5]
c = 5*ones(1,1)
D = ones(2, 3)
e = [1, 2, 3, 4, 5]
f = [1 2 3 4 5]
g = [1:5]
h = 1:5
k = 1:1:5
m = linspace(1, 5, 5)
```

```
>> clear
>> a = 5
a =
     5
>> b = [5]
b =
     5
>> c = 5*ones(1,1)
c =
     5
>> D = ones(2, 3)
D =
     1     1     1
     1     1     1
>> e = [1, 2, 3, 4, 5]
e =
     1     2     3     4     5
>> f = [1 2 3 4 5]
f =
     1     2     3     4     5
>> g = [1:5]
g =
     1     2     3     4     5
>> h = 1:5
h =
     1     2     3     4     5
>> k = 1:1:5
k =
     1     2     3     4     5
>> m = linspace(1, 5, 5)
m =
     1     2     3     4     5
```

Example02_06b.m

[6] Type the following commands (also see [7]).

```
clear
a = zeros(1,5)
a(1,5) = 8
a(5) = 9
a([1, 2, 4]) = [8, 7, 6]
a(1:4) = [2, 3, 4, 5]
[rows, cols] = size(a)
len = length(a)
b = a
c = a(1:5)
d = a(3:5)
e = a(3:length(a))
f = a(3:end)
f(5) = 10
```

```
>> clear
>> a = zeros(1,5)
a =
      0     0     0     0     0
>> a(1,5) = 8
a =
      0     0     0     0     8
>> a(5) = 9
a =
      0     0     0     0     9
>> a([1, 2, 4]) = [8, 7, 6]
a =
      8     7     0     6     9
>> a(1:4) = [2, 3, 4, 5]
a =
      2     3     4     5     9
>> [rows, cols] = size(a)
rows =
      1
cols =
      5
>> len = length(a)
len =
      5
>> b = a
b =
      2     3     4     5     9
>> c = a(1:5)
c =
      2     3     4     5     9
>> d = a(3:5)
d =
      4     5     9
>> e = a(3:length(a))
e =
      4     5     9
>> f = a(3:end)
f =
      4     5     9
>> f(5) = 10
f =
     4     5     9     0    10
```

Example02_06c.m

[11] Type the following commands (also see [12]).

```
clear
a = [1, 2; 3, 4; 5, 6]
b = 1:6
c = reshape(b, 3, 2)
d = reshape(b, 2, 3)
e = d'
c(:,3) = [7, 8, 9]
c(4,:) = [10, 11, 12]
c(4,:) = []
c(:,2:3) = []
```

```
>> clear
>> a = [1, 2; 3, 4; 5, 6]
a =
     1     2
     3     4
     5     6
>> b = 1:6
b =
     1     2    3    4    5    6
>> c = reshape(b, 3, 2)
c =
     1     4
     2     5
     3     6
>> d = reshape(b, 2, 3)
d =
     1     3     5
     2     4     6
>> e = d'
e =
     1     2
     3     4
     5     6
>> c(:,3) = [7, 8, 9]
c =
     1     4     7
     2     5     8
     3     6     9
>> c(4,:) = [10, 11, 12]
c =
     1     4     7
     2     5     8
     3     6     9
    10    11    12
>> c(4,:) = []
c =
     1     4     7
     2     5     8
     3     6     9
>> c(:,2:3) = []
c =
     1
     2
     3
```

Example02_06d.m

[14] Type the following commands (also see [15]).

```
clear
a = reshape(1:6, 3, 2)
b = [7; 8; 9]
c = horzcat(a, b)
d = [a, b]
e = b'
f = vertcat(d, e)
g = [d; e]
h = fliplr(c)
k = flipud(c)
```

```
>> clear
>> a = reshape(1:6, 3, 2)
a =
      1     4
      2     5
      3     6
>> b = [7; 8; 9]
b =
      7
      8
      9
>> c = horzcat(a, b)
c =
      1     4     7
      2     5     8
      3     6     9
>> d = [a, b]
d =
      1     4     7
      2     5     8
      3     6     9
>> e = b'
e =
      7     8     9
>> f = vertcat(d, e)
f =
      1     4     7
      2     5     8
      3     6     9
      7     8     9
>> g = [d; e]
g =
      1     4     7
      2     5     8
      3     6     9
      7     8     9
>> h = fliplr(c)
h =
      7     4     1
      8     5     2
      9     6     3
>> k = flipud(c)
k =
      3     6     9
      2     5     8
      1     4     7
```

Table 2.6a Array Creation Functions	
Function	Description
`zeros(n,m)`	Create an n-by-m matrix of all zeros
`ones(n,m)`	Create an n-by-m matrix of all ones
`eye(n)`	Create an n-by-n identity matrix
`diag(v)`	Create a square diagonal matrix with v on the diagonal
`rand(n,m)`	Create an n-by-m matrix of uniformly distributed random numbers in the interval (0,1)
`randn(n,m)`	Create an n-by-m matrix of random numbers from the standard normal distribution
`linspace(a,b,n)`	Create a row vector of n linearly spaced numbers from a to b
`[X,Y] = meshgrid(x,y)`	Create a 2-D grid coordinates based on the coordinates in vectors x and y.
Details and More: Help>MATLAB>Language Fundamentals>Matrices and Arrays	

Table 2.6b Array Replication, Concatenation, Flipping, and Reshaping	
Function	Description
`repmat(a,n,m)`	Replicate array a n times in row-dimension and m times in column-dimension
`horzcat(a,b,...)`	Concatenate arrays horizontally
`vertcat(a,b,...)`	Concatenate arrays vertically
`flipud(A)`	Flip an array upside down
`fliplr(A)`	Flip an array left-side right
`reshape(A,n,m)`	Reshape an array to an n-by-m matrix
Details and More: Help>MATLAB>Language Fundamentals>Matrices and Arrays	

2.7 Sums, Products, Minima, and Maxima

Example02_07.m

[2] Type the following commands (also see [3]).

```
clear
a = 1:5
b = sum(a)
c = cumsum(a)
d = prod(a)
e = cumprod(a)
f = diff(a)
A = reshape(1:9, 3, 3)
g = sum(A)
B = cumsum(A)
h = prod(A)
C = cumprod(A)
D = diff(A)
p = min(a)
q = max(a)
r = min(A)
s = max(A)
```

```
>> clear
>> a = 1:5
a =
     1     2     3     4     5
>> b = sum(a)
b =
    15
>> c = cumsum(a)
c =
     1     3     6    10    15
>> d = prod(a)
d =
   120
>> e = cumprod(a)
e =
     1     2     6    24   120
>> f = diff(a)
f =
     1     1     1     1
>> A = reshape(1:9, 3, 3)
A =
     1     4     7
     2     5     8
     3     6     9
>> g = sum(A)
g =
     6    15    24
>> B = cumsum(A)
B =
     1     4     7
     3     9    15
     6    15    24
>> h = prod(A)
h =
     6   120   504
>> C = cumprod(A)
C =
     1     4     7
     2    20    56
     6   120   504
>> D = diff(A)
D =
     1     1     1
     1     1     1
```

Table 2.7
Sums, Products, Minima, and Maxima

Function	Description
`sum(A)`	Sum of array elements
`cumsum(A)`	Cumulative sum
`diff(A)`	Differences between adjacent elements
`prod(A)`	Product of array elements
`cumprod(A)`	Cumulative product
`min(A)`	Minimum
`max(A)`	Maximum

```
>> p = min(a)
p =
     1
>> q = max(a)
q =
     5
>> r = min(A)
r =
     1     4     7
>> s = max(A)
s =
     3     6     9
```

2.8 Arithmetic Operators

Example02_08a.m

[2] These statements demonstrate some arithmetic operations on **matrices** (see the **Command Window** session in [3-4], next page). →

```
clear
A = reshape(1:6, 2, 3)
B = reshape(7:12, 2, 3)
C = A+B
D = A-B
E = B'
F = A*E
a = [3, 6]
b = a/F
c = b*F
G = F^2
H = A.*B
K = A./B
M = A.^2
P = A+10
Q = A-10
R = A*1.5
S = A/2
```

Table 2.8 Arithmetic Operators

Operator	Name	Description	Precedence level
`+`	`plus`	Addition	6
`-`	`minus`	Subtraction	6
`*`	`mtimes`	Multiplication	5
`/`	`mrdivide`	Division	5
`^`	`mpower`	Exponentiation	2
`.*`	`times`	Element-wise multiplication	5
`./`	`rdivide`	Element-wise division	5
`.^`	`power`	Element-wise exponentiation	2
`-`	`uminus`	Unary minus	4
`+`	`uplus`	Unary plus	4

Details and More:
Help>MATLAB>Language Fundamentals>Operators and Elementary Operations>Operator Precedence
Help>MATLAB>Language Fundamentals>Operators and Elementary Operations>Arithmetic

```
>> clear
>> A = reshape(1:6, 2, 3)
A =
      1     3     5
      2     4     6
>> B = reshape(7:12, 2, 3)
B =
      7     9    11
      8    10    12
>> C = A+B
C =
      8    12    16
     10    14    18
>> D = A-B
D =
     -6    -6    -6
     -6    -6    -6
>> E = B'
E =
      7     8
      9    10
     11    12
>> F = A*E
F =
     89    98
    116   128
>> a = [3, 6]
a =
      3     6
>> b = a/F
b =
   -13.0000   10.0000
>> c = b*F
c =
      3     6
>> G = F^2
G =
         19289       21266
         25172       27752
>> H = A.*B
H =
      7    27    55
     16    40    72
>> K = A./B
K =
     0.1429    0.3333    0.4545
     0.2500    0.4000    0.5000
>> M = A.^2
M =
      1     9    25
      4    16    36
```

```
>> P = A+10
P =
     11    13    15
     12    14    16
>> Q = A-10
Q =
     -9    -7    -5
     -8    -6    -4
>> R = A*1.5
R =
     1.5000    4.5000    7.5000
     3.0000    6.0000    9.0000
>> S = A/2
S =
     0.5000    1.5000    2.5000
     1.0000    2.0000    3.0000
```

Example02_08b.m

[8] These statements demonstrate some arithmetic operations on **vectors** (also see [9]). Remember that a vector is a special case of matrices. Thus operations on vectors are special cases of those on matrices.

```
clear
a = 1:4
b = 5:8
c = a+b
d = a-b
e = a*(b')
f = (a')*b
g = a/b
h = a.*b
k = a./b
m = a.^2
```

```
>> clear
>> a = 1:4
a =
     1     2     3     4
>> b = 5:8
b =
     5     6     7     8
>> c = a+b
c =
     6     8    10    12
>> d = a-b
d =
    -4    -4    -4    -4
>> e = a*(b')
e =
    70
>> f = (a')*b
f =
     5     6     7     8
    10    12    14    16
    15    18    21    24
    20    24    28    32
>> g = a/b
g =
    0.4023
>> h = a.*b
h =
     5    12    21    32
>> k = a./b
k =
    0.2000    0.3333    0.4286    0.5000
>> m = a.^2
m =
     1     4     9    16
```

Example02_08c.m

[11] These statements demonstrate some arithmetic operations on **scalars** (also see [12]). Remember that a scalar is a special case of matrices. Thus operations on scalar are special cases of those on matrices.

```
clear
a = 6
b = 4
c = a+b
d = a-b
e = a*b
f = a/b
g = a^2
h = a.*b
k = a./b
m = a.^2
```

```
>> clear
>> a = 6
a =
     6
>> b = 4
b =
     4
>> c = a+b
c =
    10
>> d = a-b
d =
     2
>> e = a*b
e =
    24
>> f = a/b
f =
    1.5000
>> g = a^2
g =
    36
>> h = a.*b
h =
    24
>> k = a./b
k =
    1.5000
>> m = a.^2
m =
    36
```

2.9 Relational and Logical Operators

Table 2.9a Relational Operators

Operator	Description	Precedence level
`==`	Equal to	8
`~=`	No equal to	8
`>`	Greater than	8
`<`	Less than	8
`>=`	Greater than or equal to	8
`<=`	Less than or equal to	8
`isequal`	Determine array equality	

Details and More: Help>MATLAB>Language Fundamentals> Operators and Elementary Operations>Relational Operations

Table 2.9b Logical Operators

Operator	Description	Precedence level
`&`	Logical AND	9
`|`	Logical OR	10
`~`	Logical NOT	4
`&&`	Logical AND (short-circuit)	11
`||`	Logical OR (short-circuit)	12

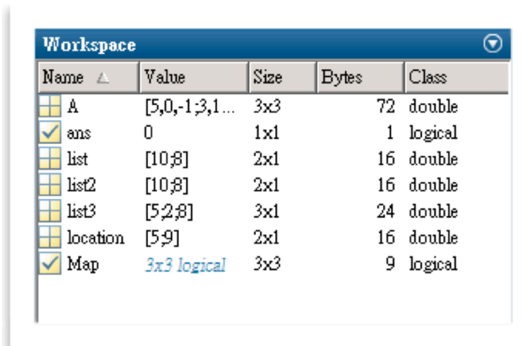
Details andMore: Help>MATLAB>Language Fundamentals> Operators and Elementary Operations>Logical Operations

Example02_09.m

[2] These statements demonstrate some relational and logical operations (also see [3, 4], next page). →

```
clear
A = [5,0,-1; 3,10,2; 0,-4,8]
Map = (A > 6)
location = find(Map)
list = A(location)
list2 = A(find(A>6))
list3 = A(find(A>0 & A<=8 & A~=3))
find(A)'
~A
~~A
isequal(A, ~~A)
```

```
clear
A = [5,0,-1; 3,10,2; 0,-4,8]
A =
     5     0    -1
     3    10     2
     0    -4     8
Map = (A > 6)
Map =
  3×3 logical array
   0   0   0
   0   1   0
   0   0   1
location = find(Map)
location =
     5
     9
list = A(location)
list =
    10
     8
list2 = A(find(A>6))
list2 =
    10
     8
list3 = A(find(A>0 & A<=8 & A~=3))
list3 =
     5
     2
     8
find(A)'
ans =
    1     2     5     6     7     8     9
~A
ans =
 3×3 logical array
   0   1   0
   0   0   0
   1   0   0
~~A
ans =
  3×3 logical array
   1   0   1
   1   1   1
   0   1   1
isequal(A, ~~A)
ans =
 logical
   0
```



Name	Value	Size	Bytes	Class
A	[5,0,-1;3,1...	3x3	72	double
ans	0	1x1	1	logical
list	[10;8]	2x1	16	double
list2	[10;8]	2x1	16	double
list3	[5;2;8]	3x1	24	double
location	[5;9]	2x1	16	double
Map	3x3 logical	3x3	9	logical

2.10 String Manipulations

Example02_10a.m: String Manipulations

[2] Type and run the following statements, which demonstrate some string manipulations. Input your name and age as shown in [3].

```
clear
a = 'Hello,';
b = 'world!';
c = [a, ' ', b];
disp(c)
name = input('What is your name? ', 's');
years = input('What is your age? ');
disp(['Hello, ', name, '! You are ', num2str(years), ' years old.'])
str = sprintf('Pi = %.8f', pi);
disp(str)
Names1 = [
    'David  '
    'John   '
    'Stephen'];
Names2 = char('David', 'John', 'Stephen');
if isequal(Names1, Names2)
    disp('The two lists are equal.')
end
name = deblank(Names1(2,:));
disp(['The name ', name, ' has ', num2str(length(name)), ' characters.'])
```

```
>> Example02_10a
Hello, world!
What is your name? Lee
What is your age? 60
Hello, Lee! You are 60 years old.
Pi = 3.14159265
The two lists are equal.
The name John has 4 characters.
>>
```

Example02_10b.m: A Simple Calculator

[6] This program uses function `eval` to create a simple calculator. Type and run the program (see a test run in [7]).

```
clear
disp('A Simple Calculator')
while true
    expr = input('Enter an expression (or quit): ', 's');
    if strcmp(expr,'quit')
        break
    end
    disp([expr, ' = ', num2str(eval(expr))])
end
```

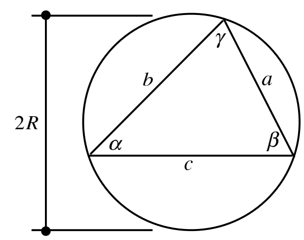
```
>> Example02_10b
A Simple Calculator
Enter an expression (or quit): 3+5
3+5 = 8
Enter an expression (or quit): sin(pi/4) + (2 + 2.1^2)*3
sin(pi/4) + (2 + 2.1^2)*3 = 19.9371
Enter an expression (or quit): quit
>>
```

Table 2.10 String Manipulations

Function	Description
`A = char(a,b,...)`	Convert the strings to a matrix of rows of the strings, padding blanks
`disp(X)`	Display value of variable
`x = input(prompt,'s')`	Request user input
`s = sprintf(format,a,b,...)`	Write formatted data to a string
`s = num2str(x)`	Convert number to string
`x = str2num(s)`	Convert string to number
`x = str2double(s)`	Convert string to double precision value
`x = eval(exp)`	Evaluate a MATLAB expression
`s = deblank(str)`	Remove trailing blanks from a string
`s = strtrim(str)`	Remove leading and trailing blanks from a string
`tf = strcmp(s1,s2)`	Compare two strings (case sensitive)
`tf = strcmpi(s1,s2)`	Compare two strings (case insensitive)

Details and More:
Help>MATLAB>Language Fundamentals>Data Types>Characters and Strings; Data Type Conversion

2.11 Expressions



Example02_11a.m: Law of Sines

[3] This script calculates the three angles α, β, γ of a triangle and its area A, given three sides $a = 5$, $b = 6$, and $c = 7$. →

```
clear
a = 5;
b = 6;
c = 7;
R = a*b*c/sqrt((a+b+c)*(a-b+c)*(b-c+a)*(c-a+b))
alpha = asind(a/(2*R))
beta = asind(b/(2*R))
gamma = asind(c/(2*R))
sumAngle = alpha + beta + gamma
A1 = a*b*c/(4*R)
A2 = b*c*sind(alpha)/2
```

```
>> Example02_11a
R =
     3.5722
alpha =
   44.4153
beta =
   57.1217
gamma =
   78.4630
sumAngle =
   180
A1 =
   14.6969
A2 =
   14.6969
>>
```

Example02_11b.m: Law of Cosines

[6] The law of cosines states that (see *Wikipedia>Trigonometry*; use the same notations in [2], last page):

$$a^2 = b^2 + c^2 - 2bc\cos\alpha \quad \text{or} \quad \alpha = \cos^{-1}\frac{b^2 + c^2 - a^2}{2bc}$$

With $a = 5$, $b = 6$, $c = 7$, the angle α, β, and γ can be calculated as follows:

```
clear
a = 5; b = 6; c = 7;
alpha = acosd((b^2+c^2-a^2)/(2*b*c))
beta = acosd((c^2+a^2-b^2)/(2*c*a))
gamma = acosd((a^2+b^2-c^2)/(2*a*b))
```

```
>> Example02_11b
alpha =
   44.4153
beta =
   57.1217
gamma =
   78.4630
>>
```

Table 2.11a
Special Characters

Special characters	Description
`[ ]`	Brackets
`{ }`	Braces
`( )`	Parentheses
`'`	Matrix transpose
`.`	Field access
`...`	Continuation
`,`	Comma
`;`	Semicolon
`:`	Colon
`@`	Function handle

Details and More:
Help>MATLAB>Language Fundamentals>Operators and Elementary Operations>MATLAB Operators and Special Characters

Table 2.11b
Elementary Math Functions

Function	Description
`sin(x)`	Sine (in radians)
`sind(x)`	Sine (in degrees)
`asin(x)`	Inverse sine (in radians)
`asind(x)`	Inverse sine (in degrees)
`cos(x)`	Cosine (in radians)
`cosd(x)`	Cosine (in degrees)
`acos(x)`	Inverse cosine (in radians)
`acosd(x)`	Inverse cosine (in degrees)
`tan(x)`	Tangent (in radians)
`tand(x)`	Tangent (in degrees)
`atan(x)`	Inverse tangent (in radians)
`atand(x)`	Inverse tangent (in degrees)
`atan2(y,x)`	Four-quadrant inverse tangent (radians)
`atan2d(y,x)`	Four-quadrant inverse tangent (degrees)
`abs(x)`	Absolute value
`sqrt(x)`	Square root
`exp(x)`	Exponential (base *e*)
`log(x)`	Logarithm (base *e*)
`log10(x)`	Logarithm (base 10)
`factorial(n)`	Factorial
`sign(x)`	Sign of a number
`rem(a,b)`	Remainder after division
`mod(a,m)`	Modulo operation

Details and More:
Help>MATLAB>Mathematics>Elementary Math

2.12 Example: Function Approximation

Example02_12a.m: Scalar Expressions

[2] This script calculates the value of $\sin(\pi/4)$ using the Taylor series in Eq. (a). The screen output is shown in [3].

```
clear
x = pi/4;
term1 = x;
term2 = -x^3/(3*2);
term3 = x^5/(5*4*3*2);
term4 = -x^7/(7*6*5*4*3*2);
format long
sinx =term1+term2+term3+term4
exact = sin(x)
error = (sinx-exact)/exact
```

```
sinx =
   0.707106469575178
exact =
   0.707106781186547
error =
    -4.406850247592559e-07
```

Example02_12b.m: Use of For-Loop

[6] Type and run this program. The screen output is the same as that of Example02_12a.m ([3], last page).

```
17   clear
18   x = pi/4; n = 4; sinx = 0;
19   for k = 1:n
20       sinx = sinx + ((-1)^(k-1))*(x^(2*k-1))/factorial(2*k-1);
21   end
22   format long
23   sinx
24   exact = sin(x)
25   error = (sinx-exact)/exact
```

```
sinx =
   0.707106781179619
exact =
   0.707106781186547
error =
    -9.797690960678494e-12
```

Example02_12c.m: Vector Expressions

[9] This script produces the same output as that of Example02_12b.m ([3], last page) using a vector expression (line 29) in place of the `for`-loop used in Example02_12b.m (lines 19-21). →

```
26   clear
27   x = pi/4; n = 4; k = 1:n;
28   format long
29   sinx = sum(((-1).^(k-1)).*(x.^(2*k-1))./factorial(2*k-1))
30   exact = sin(x)
31   error = (sinx-exact)/exact
```

About Example02_12c.m

[10] In line 27, the variable `k` is created as a row vector of four elements; `k = [1,2,3,4]`. The `for`-loop (lines 19-21) is now replaced by a vector expression (line 29), which uses the function `sum` and element-wise operators (`.^`, `.*`, and `./`). To help you understand the statement in line 29, we break it into several steps:

```
step1 = k-1
step2 = (-1).^step1
step3 = 2*k-1
step4 = x.^step3
step5 = step2.*step4
step6 = factorial(step3)
step7 = step5./step6
step8 = sum(step7)
sinx = step8
```

Using `k = [1,2,3,4]`, following the descriptions of element-wise operations (2.8[6-7], pages 95-96) and the function `sum` for vectors (2.7[5], page 90), we may further decode these steps as follows:

```
step1 = [0,1,2,3]
Step2 = (-1).^[0,1,2,3] ≡ [1,-1,1,-1]
step3 = [1,3,5,7]
step4 = x.^[1,3,5,7] ≡ [x,x^3,x^5,x^7]
step5 = [1,-1,1,-1].*[x,x^3,x^5,x^7] ≡ [x,-x^3,x^5,-x^7]
step6 = factorial([1,3,5,7]) ≡ [1,6,120,5040]
step7 = [x,-x^3,x^5,-x^7]./[1,6,120,5040] ≡ [x,-x^3/6,x^5/120,-x^7/5040]
step8 = x-x^3/6+x^5/120-x^7/5040
sinx = x-x^3/6+x^5/120-x^7/5040
```

Substituting `x` with `pi/4`, we have `sinx = 0.707106469575178`.

Example02_12d.m: Matrix Expressions

[11] This script calculates sin(x) for various x values and produces a graph as shown in [12], next page. A matrix expression (line 37) is used in this script. →

```
32  clear
33  x = linspace(0,pi/2,20);
34  n = 4;
35  k = 1:n;
36  [X, K] = meshgrid(x, k);
37  sinx = sum(((-1).^(K-1)).*(X .^ (2*K-1))./factorial(2*K-1));
38  plot(x*180/pi, sinx, 'o', x*180/pi, sin(x))
39  title('Approximation of sin(x)')
40  xlabel('x (deg)')
41  ylabel('sin(x) (dimensionless)')
42  legend('Approximation', 'Exact', 'Location', 'southeast')
```

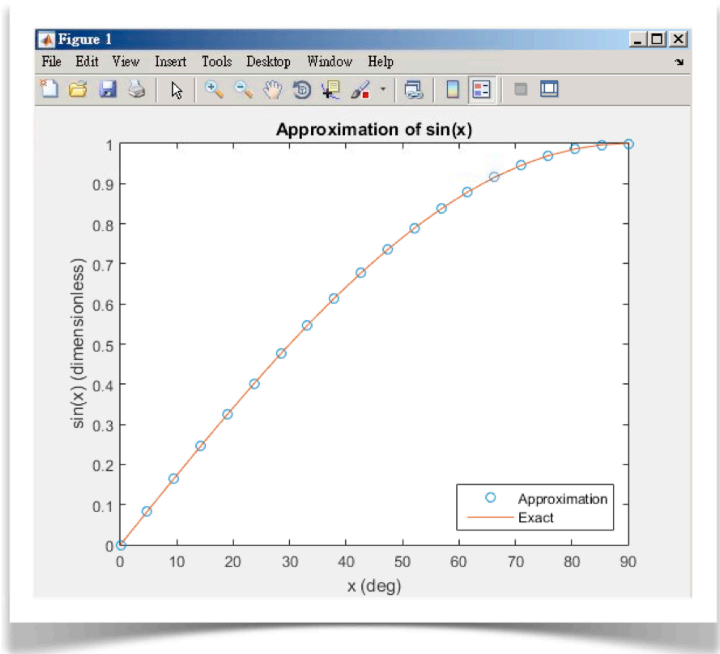
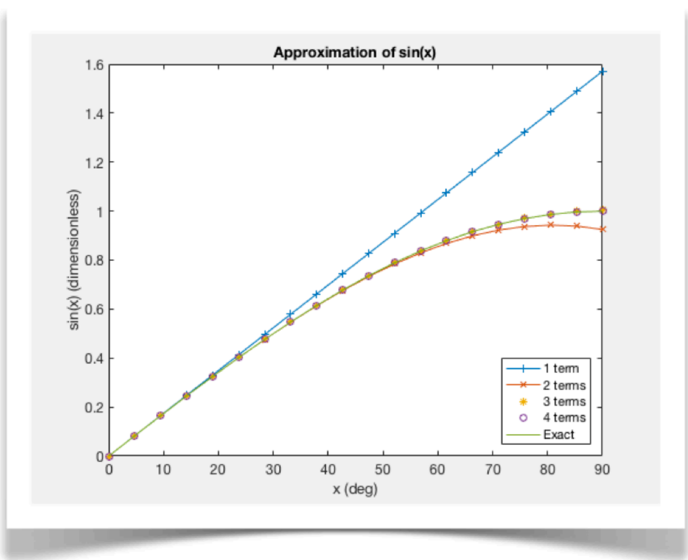


Figure 1
File Edit View Insert Tools Desktop Window Help
Approximation of sin(x)
sin(x) (dimensionless)
x (deg)
Approximation
Exact
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
10
20
30
40
50
60
70
80
90

Example02_12e.m: Multiple Curves

[14] This script plots four approximated curves and an exact curve of sin(x) as shown in [15], the four approximated curves corresponding to the Taylor series of 1, 2, 3, and 4 items, respectively.

```
clear
x = linspace(0,pi/2,20);
n = 4;
k = (1:n);
[X, K] = meshgrid(x, k);
sinx = cumsum(((-1).^(K-1)).*(X .^ (2*K-1))./factorial(2*K-1));
plot(x*180/pi, sinx(1,:), '+-', ...
     x*180/pi, sinx(2,:), 'x-', ...
     x*180/pi, sinx(3,:), '*', ...
     x*180/pi, sinx(4,:), 'o', ...
     x*180/pi, sin(x))
title('Approximation of sin(x)')
xlabel('x (deg)')
ylabel('sin(x) (dimensionless)')
legend('1 term', '2 terms', '3 terms', '4 terms', 'Exact', ...
       'Location', 'southeast')
```

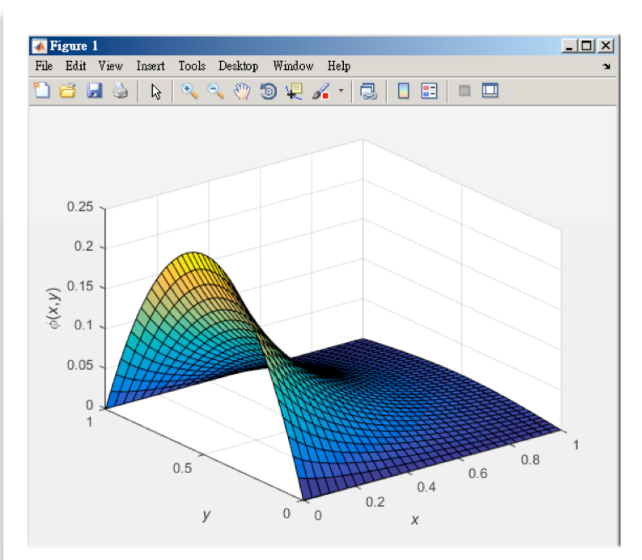


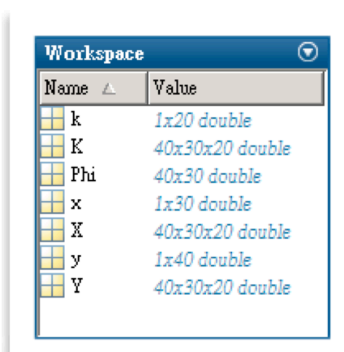
2.13 Example: Series Solution of a Laplace Equation

Example02_13.m: Series Solution of a Laplace Equation

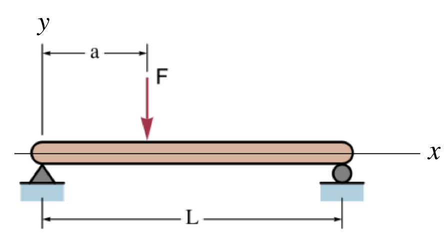
[2] This script calculates the solution $\phi(x,y)$ according to Eq. (a) and plots a three-dimensional surface $\phi = \phi(x,y)$ [3].

```
clear
k = 1:20;
x = linspace(0,1,30);
y = linspace(0,1,40);
[X,Y,K]  = meshgrid(x, y, k);
Phi = sum(4*(1-cos(K*pi))./(K*pi).^3.*exp(-K.*X*pi).*sin(K.*Y*pi), 3);
surf(x, y, Phi)
xlabel('\itx')
ylabel('\ity')
zlabel('\phi(\itx\rm,\ity\rm)')
```





2.14 Example: Deflection of Beams

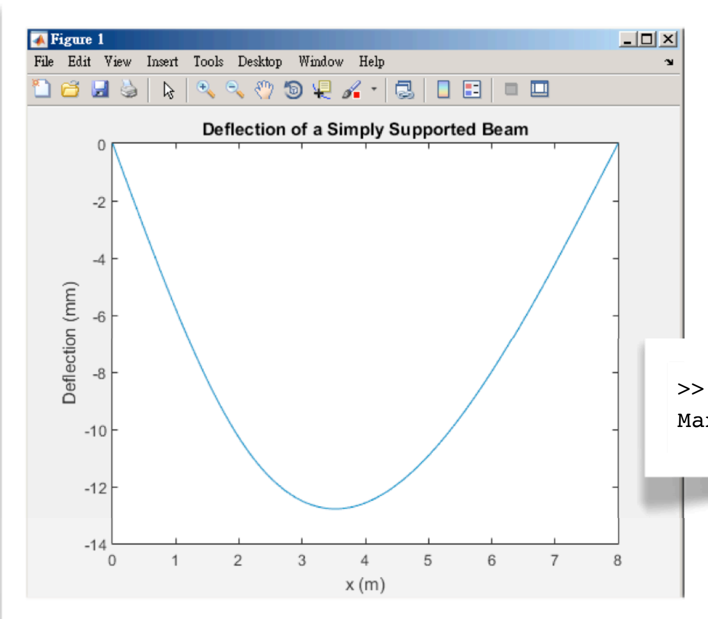


[2] This is the simply supported beam considered in this section.

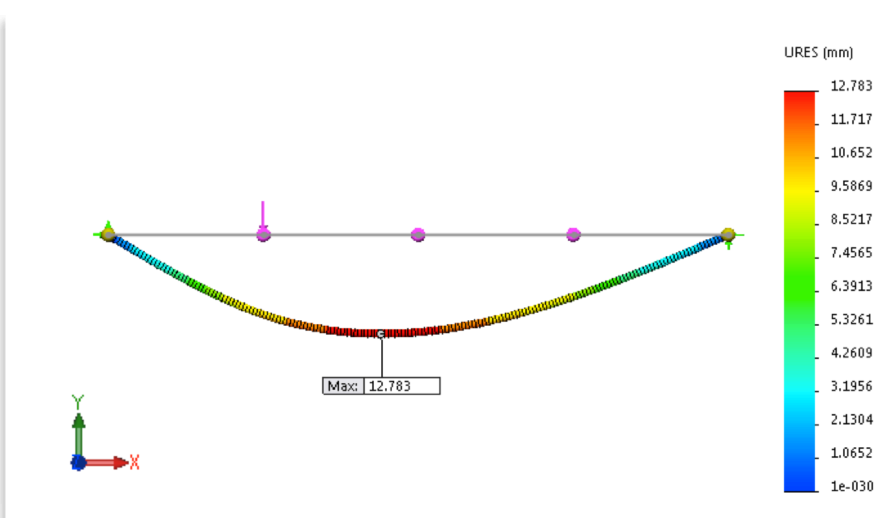
Example02_14.m: Deflection of Beams

[3] This script produces a graphic output as shown in [4] (next page) and a text output as shown in [5] (next page). →

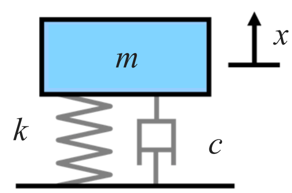
```
clear
w = 0.1;
h = 0.1;
L = 8;
E = 210e9;
F = 3000;
a = L/4;
I = w*h^3/12;
R = F/L*(L-a);
theta = F*a/(6*E*I*L)*(2*L-a)*(L-a);
x = linspace(0,L,100);
y = -theta*x+R*x.^3/(6*E*I)-F/(6*E*I)*((x>a).*((x-a).^3));
plot(x,y*1000)
title('Deflection of a Simply Supported Beam')
xlabel('x (m)'); ylabel('Deflection (mm)')
y = -y;
[ymax, index] = max(y);
fprintf('Maximum deflection %.2f mm at x = %.2f m\n', ymax*1000, x(index))
```

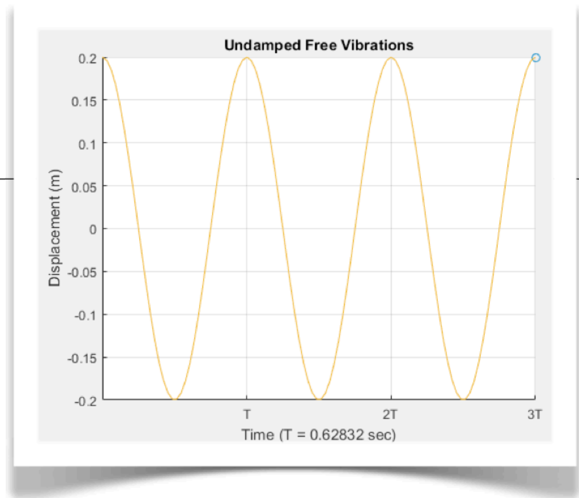



```
>> Example02_14
Maximum deflection 12.78 mm at x = 3.56 m
```



2.15 Example: Vibrations of Supported Machines

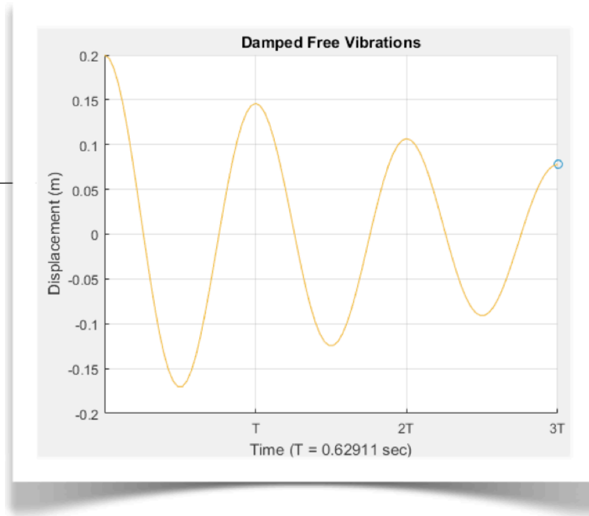




Example02_15a.m: Undamped Free Vibrations

[4] This program calculates and plots the solution $x(t)$ in Eqs. (b, c), last page. The graphic output is shown in [5].

```
clear
m = 1; k = 100; delta = 0.2;
omega = sqrt(k/m);
T = 2*pi/omega;
t = linspace(0, 3*T, 100);
x = delta*cos(omega*t);
axes('XTick', T:T:3*T, 'XTickLabel', {'T','2T','3T'});
axis([0, 3*T, -0.2, 0.2])
grid on
hold on
comet(t, x)
title('Undamped Free Vibrations')
xlabel(['Time (T = ', num2str(T), ' sec)'])
ylabel('Displacement (m)')
```



Example02_15b.m: Damped Free Vibrations

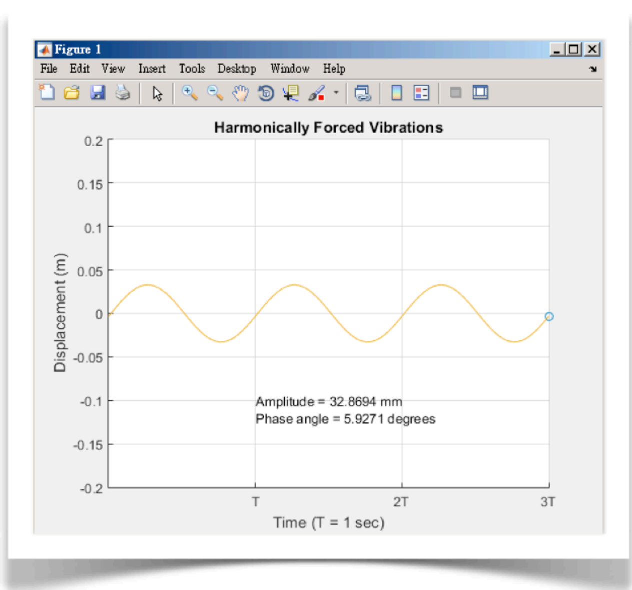
[7] This program calculates and plots the solution $x(t)$ in Eqs. (f, g), last page. The graphic output is shown in [8].

```
clear
m = 1; k = 100; c = 1; delta = 0.2;
omega = sqrt(k/m);
cC = 2*m*omega;
omegaD = omega*sqrt(1-(c/cC)^2);
T = 2*pi/omegaD;
t = linspace(0, 3*T, 100);
x = delta*exp(-c*t/(2*m)).*(cos(omegaD*t)+c/(2*m*omegaD)*sin(omegaD*t));
axes('XTick', T:T:3*T, 'XTickLabel', {'T','2T','3T'});
axis([0, 3*T, -0.2, 0.2])
grid on
hold on
comet(t, x)
title('Damped Free Vibrations')
xlabel(['Time (T = ', num2str(T), ' sec)'])
ylabel('Displacement (m)')
```

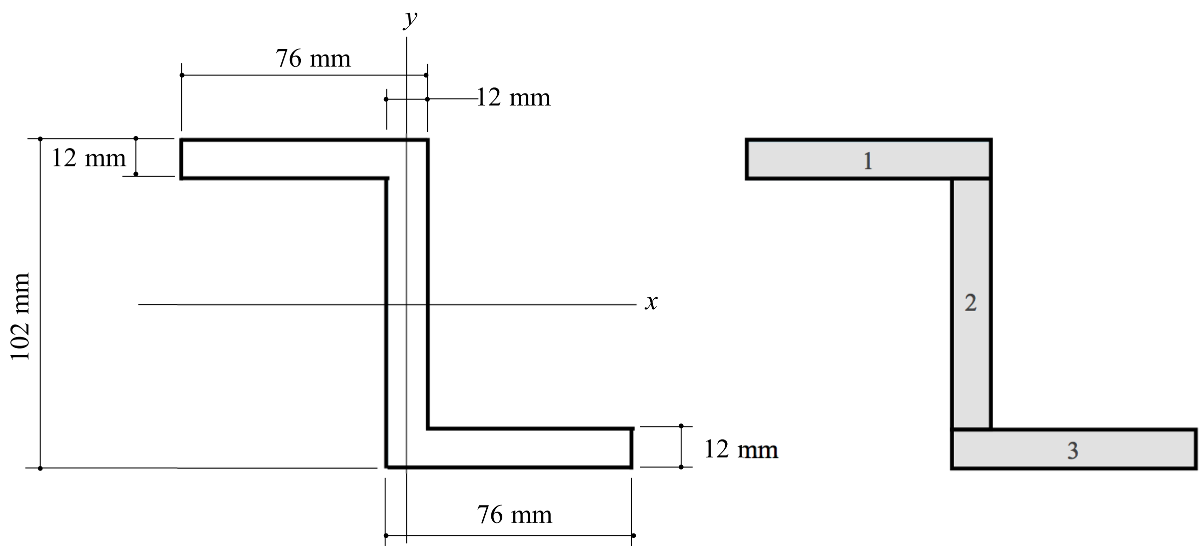
Example02_15c.m: Forced Vibrations

[10] This program plots the steady-state response $x(t)$ in Eqs. (i, j, k), last page. The graphic output is shown in [11].

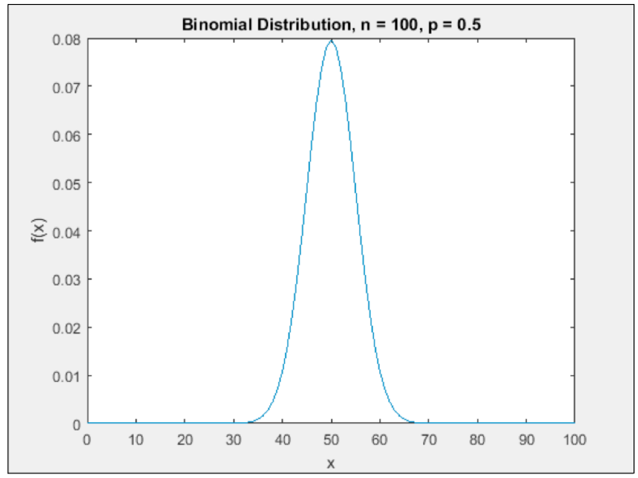
```
clear
% System parameters
m = 1; k = 100; c = 1;
f = 2; omegaF = 2*pi;

% System response
omega = sqrt(k/m);
cC = 2*m*omega;
rC = c/cC;
rW = omegaF/omega;
xm = (f/k)/sqrt((1-rW^2)^2+(2*rC*rW)^2);
phi = atan((2*rC*rW)/(1-rW^2));
T = 2*pi/omegaF;
t = linspace(0, 3*T, 100);
x = xm*sin(omegaF*t-phi);

% Graphic output
axes('XTick', T:T:3*T, 'XTickLabel', {'T','2T','3T'});
axis([0, 3*T, -0.2, 0.2])
grid on
hold on
comet(t, x)
title('Harmonically Forced Vibrations')
xlabel(['Time (T = ', num2str(T), ' sec)'])
ylabel('Displacement (m)')
text(T,-0.1,['Amplitude = ', num2str(xm*1000), ' mm'])
text(T,-0.12,['Phase angle = ', num2str(phi*180/pi), ' degrees'])
```

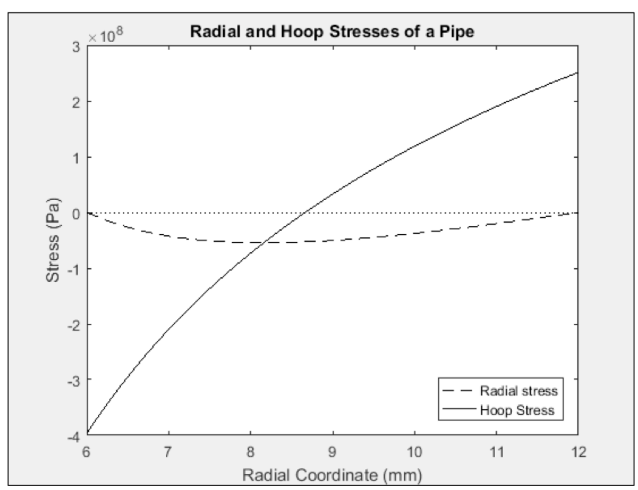


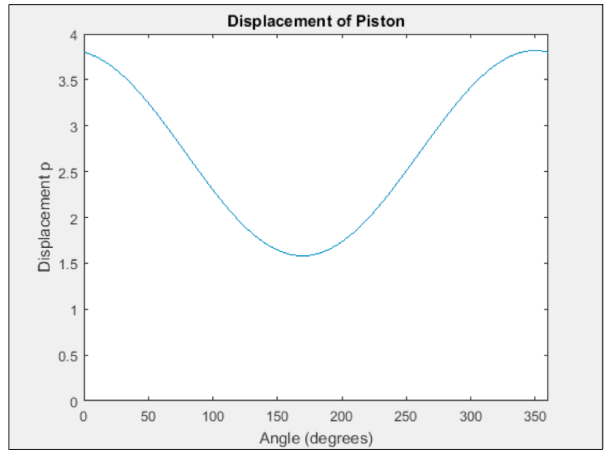
2.16 Additional Exercise Problems



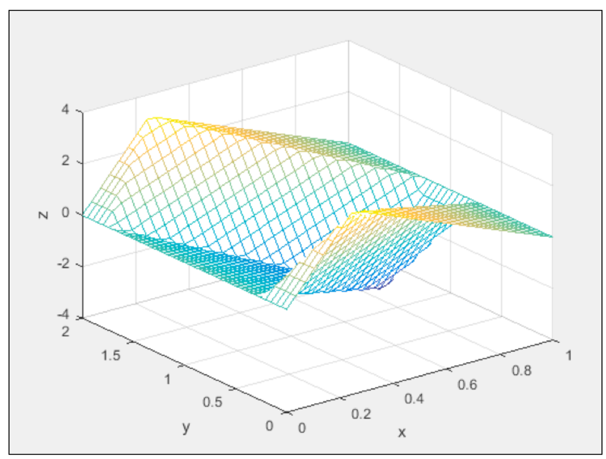
Rectangle	b mm	h mm	$\bar{x}$ mm	$\bar{y}$ mm	$\bar{I}_x$ mm^4	$\bar{I}_y$ mm^4	A mm^2	$A\bar{y}^2$ mm^4	$A\bar{x}^2$ mm^4	I_x mm^4	I_y mm^4
1	76	12	-32	45	10944	438976	912	1846800	933888	1857744	1372864
2	12	78	0	0	474552	11232	936	0	0	474552	11232
3	76	12	32	-45	10944	438976	912	1846800	933888	1857744	1372864
Total							2760			4190040	2756960
Notes: $\bar{I}_x = bh^3/12$, $\bar{I}_y = hb^3/12$, $A = bh$, $I_x = \bar{I}_x + A\bar{y}^2$, $I_y = \bar{I}_y + A\bar{x}^2$											


Binomial Distribution, n = 100, p = 0.5
f(x)
x
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0
10
20
30
40
50
60
70
80
90
100


Radial and Hoop Stresses of a Pipe
× 10^8
Stress (Pa)
Radial Coordinate (mm)
Radial stress
Hoop Stress
3
2
1
0
-1
-2
-3
-4
6
7
8
9
10
11
12



Displacement of Piston
Displacement p
Angle (degrees)
0
0.5
1
1.5
2
2.5
3
3.5
4
0
50
100
150
200
250
300
350



z
y
x
4
2
0
-2
-4
2
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1

Chapter 3

Flow Controls, Functions, and Programs

A program may consist of one or more program files; one of them is the main program file. Each program file may consist of a main program and functions. The first function, if exists, in a program file is called the main function; others are called subfunctions, nested functions, or anonymous functions. Subfunctions are local to the program file to which they belong, while nested functions and anonymous functions are local to the function to which they belong. Execution of a program starts from the main program (or main function) of the main program file.

3.1 If-Blocks

Example03_01.m: If-Blocks

```
clear
n1 = input('Enter a number: ');
n2 = input('Enter another number: ');
disp('Test #1')
string = 'At least one is non-positive';
if n1>0 && n2>0
    string = 'Both are positive';
end
disp(string)

disp('Test #2')
if n1>0 && n2>0
    disp('Both are positive.')
else
    disp('At least one is non-positive.')
end

disp('Test #3')
if n1>0 && n2>0
    disp('Both are positive.')
elseif n1==0 || n2 == 0
    disp('At least one is zero.')
else
    disp('At least one is negative.')
end

disp('Test #4')
if n1>0 && n2>0
    disp('Both are positive.')
elseif n1==0 || n2 == 0
    disp('At least one is zero.')
elseif n1*n2 < 0
    disp('They have opposite signs.')
else
    disp('Both are negative')
end

disp('Test #5')
a = [n1, n2];
if all(a>0)
    disp('Both are positive')
elseif any(a>0)
    disp('One of them is positive.')
else
    disp('None of them is positive.')
end
```

```
>> Example03_01
Enter a number: 5
Enter another number: -2
Test #1
At least one is non-positive
Test #2
At least one is non-positive.
Test #3
At least one is negative.
Test #4
They have opposite signs.
Test #5
One of them is positive.
>>
```

3.2 Switch-Blocks

Example03_02a.m: Pizza Menu

[2] Type and run this program. See [3] for an input/output.

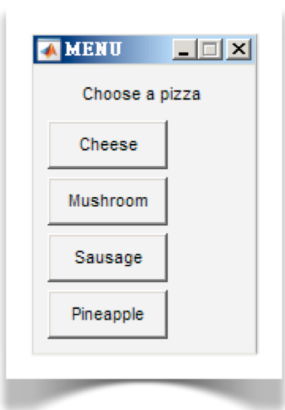
```
clear
fprintf(['Cheese\n' ...
    'Mushroom\n' ...
    'Sausage\n' ...
    'Pineapple\n'])
choice = input('Choose a pizza: ', 's');
choice = lower(strtrim(choice));
switch choice
    case 'cheese'
        disp('Cheese pizza $3.99')
    case 'mushroom'
        disp('Mushroom pizza $3.66')
    case 'sausage'
        disp('Sausage pizza $4.22')
    case 'pineapple'
        disp('Pineapple pizza $2.99')
    otherwise
        disp('Sorry?')
end
```

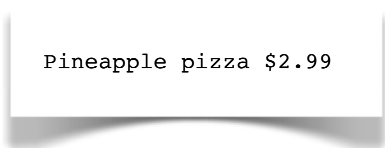
```
Cheese
Mushroom
Sausage
Pineapple
Choose a pizza: pineapple
Pineapple pizza $2.99
```

Example03_02b.m: Pizza Menu

[5] Type and run this program. See [6, 7] for an input/output.

```
clear
choice = menu('Choose a pizza', ...
    'Cheese', 'Mushroom', 'Sausage', 'Pineapple');
switch choice
    case 1
        disp('Cheese pizza $3.99')
    case 2
        disp('Mushroom pizza $3.66')
    case 3
        disp('Sausage pizza $4.22')
    case 4
        disp('Pineapple pizza $2.99')
end
```



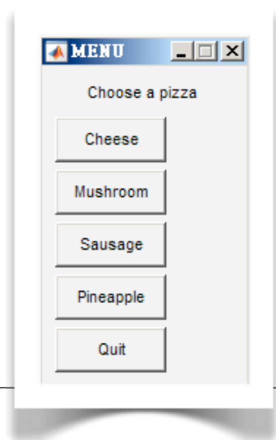


3.3 While-Loops

Example03_03a.m: Pizza Menu

[2] Type and run this program. See [3, 4] for an input/output.

```
clear
choice = 0;
while choice ~= 5
    choice = menu('Choose a pizza', ...
        'Cheese', 'Mushroom', 'Sausage', 'Pineapple', 'Quit');
    switch choice
        case 1
            disp('Cheese pizza $3.99')
        case 2
            disp('Mushroom pizza $3.66')
        case 3
            disp('Sausage pizza $4.22')
        case 4
            disp('Pineapple pizza $2.99')
        case 5
            disp('Bye!')
    end
end
```




```
Sausage pizza $4.22
Mushroom pizza $3.66
Cheese pizza $3.99
Pineapple pizza $2.99
Bye!
```

Example03_03b.m: Pizza Menu

[5] Program Example03_03a.m can be slightly modified using a "forever-true statement" (line 20) and a `break`-statement (line 34). I personally prefer this style to that in Example03_03a.m.

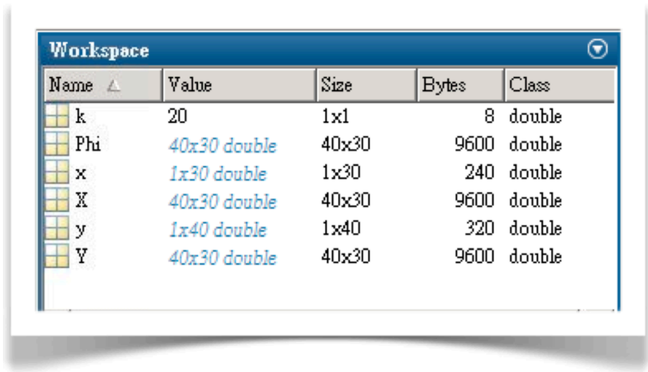
```
19  clear
20  while 1
21      choice = menu('Choose a pizza', ...
22          'Cheese', 'Mushroom', 'Sausage', 'Pineapple', 'Quit');
23      switch choice
24          case 1
25              disp('Cheese pizza $3.99')
26          case 2
27              disp('Mushroom pizza $3.66')
28          case 3
29              disp('Sausage pizza $4.22')
30          case 4
31              disp('Pineapple pizza $2.99')
32          case 5
33              disp('Bye!')
34              break;
35      end
36  end
```

3.4 For-Loops

Example03_04a.m: For-Loops

[2] The program Example02_13.m (page 113) is rewritten using a `for`-loop as follows. Type and test-run the program. The output is the same as before (2.13[3], page 113); this program uses less memory space (you may compare [3], next page, with 2.13[4], page 113) but takes more computing time. →

```
clear
x = linspace(0,1,30);
y = linspace(0,1,40);
[X,Y]  = meshgrid(x, y);
Phi = zeros(40,30);
for k = 1:20
    Phi = Phi+4*(1-cos(k*pi))/(k*pi)^3*exp(-k*X*pi).*sin(k*Y*pi);
end
surf(x, y, Phi)
xlabel('\itx')
ylabel('\ity')
zlabel('\phi(\itx\rm,\ity\rm)')
```



Workspace

Name	Value	Size	Bytes	Class
k	20	1x1	8	double
Phi	40x30 double	40x30	9600	double
x	1x30 double	1x30	240	double
X	40x30 double	40x30	9600	double
y	1x40 double	1x40	320	double
Y	40x30 double	40x30	9600	double

Example03_04b.m: Nested For-Loops

[5] The *`statements`* inside a `for`-loop ([1], last page) may include other `for`-loops, called **nested loops**. Type and run the following program. There are 3 layers of `for`-loops. The expression in lines 21-22 is now a scalar expression. The output is the same as before (2.13[3], page 113); the **Workspace** is the same as [3] except that additional two variables (`i` and `j`) are added to the **Workspace**. →

```
13  clear
14  x = linspace(0,1,30);
15  y = linspace(0,1,40);
16  [X,Y]  = meshgrid(x, y);
17  Phi = zeros(40,30);
18  for i = 1:40
19      for j = 1:30
20          for k = 1:20
21              Phi(i,j) = Phi(i,j)+4*(1-cos(k*pi))/(k*pi)^3 ...
22                  *exp(-k*X(i,j)*pi)*sin(k*Y(i,j)*pi);
23          end
24      end
25  end
26  surf(x, y, Phi)
27  xlabel('\itx')
28  ylabel('\ity')
29  zlabel('\phi(\itx\rm,\ity\rm)')
```

Measuring Time: Functions `tic` and `toc`

[6] To measure the time needed to execute a portion of a program, we may insert the function `tic` (which starts a stopwatch) before the portion and insert the function `toc` (which stops the stopwatch) after the portion. MATLAB will report on the **Command Window** the **elapsed time** (in seconds) to execute the portion of program. For example, insert the functions `tic` and `toc` in the program Example03_04b.m as follows:

```
tic
for i = 1:40
    for j = 1:30
        for k = 1:20
            Phi(i,j) = Phi(i,j)+4*(1-cos(k*pi))/(k*pi)^3 ...
                *exp(-k*X(i,j)*pi)*sin(k*Y(i,j)*pi);
        end
    end
end
toc
```

To compare with program Example02_13.m, we also insert functions `tic` and `toc` as follows

```
tic
Phi = sum(4*(1-cos(K*pi))./(K*pi).^3.*exp(-K.*X*pi).*sin(K.*Y*pi), 3);
toc
```

It leaves you to confirm that, as mentioned in [4], last page, using built-in array operation capabilities (Example02_13.m, page 113) is usually computationally efficient than using `for`-loops (Example03_04b.m, last page). Further, programs using built-in array operation capabilities can be easily adapted to a parallel computing environment. #

3.5 User-Defined Functions

Example03_05a.m: User-Defined Functions

[1] Type the following program, which calculates the period and the response of a damped free vibration system based on the formula in Eqs. (f, g), page 118; click **Save** button (1.5[6], page 22) and accept the default file name Example03_05a, which is the same as the function name (line 1). You must accept the default name; the file name must match the main function name in a program file. MATLAB saves the file as Example03_05a.m. Note that a file starting with a function cannot be saved as a **Live Script**.

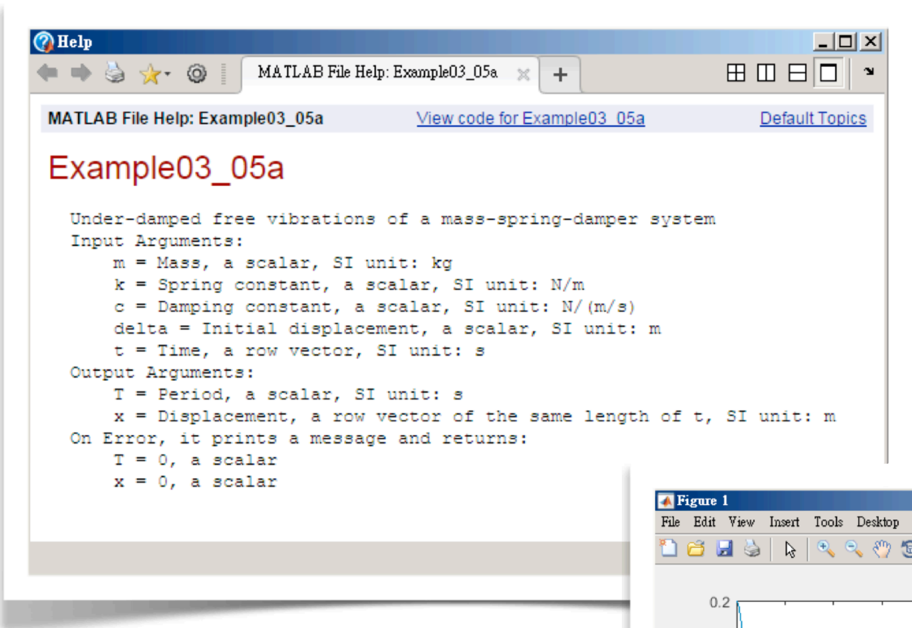
You cannot run this function by clicking the **Run** button (1.5[8], page 22). You must "call" the function and provide its input arguments from the **Command Window** (as demonstrated in [2]), from a **script**, or from a **Live Script**.

```
 1  function [T, x] = Example03_05a(m, k, c, delta, t)
 2  % Under-damped free vibrations of a mass-spring-damper system
 3  % Input Arguments:
 4  %     m = Mass, a scalar, SI unit: kg
 5  %     k = Spring constant, a scalar, SI unit: N/m
 6  %     c = Damping constant, a scalar, SI unit: N/(m/s)
 7  %     delta = Initial displacement, a scalar, SI unit: m
 8  %     t = Time, a row vector, SI unit: s
 9  % Output Arguments:
10  %     T = Period, a scalar, SI unit: s
11  %     x = Displacement, a row vector of the same length of t, SI unit: m
12  % On Error, it prints a message and returns:
13  %     T = 0, a scalar
14  %     x = 0, a scalar
15  omega = sqrt(k/m);
16  cC = 2*m*omega;
17  if c>= cC
18      disp('Not an under-damped system!')
19      T = 0; x = 0;
20      return;
21  end
22  omegaD = omega*sqrt(1-(c/cC)^2);
23  T = 2*pi/omegaD;
24  x = delta*exp(-c*t/(2*m)).*(cos(omegaD*t)+c/(2*m*omegaD)*sin(omegaD*t));
25  end
```

Example03_05b.m: Calling Functions

[2] On the **Command Window**, type the following commands. →

```
26  >> doc Example03_05a
27  >> time = linspace(0,5,100);
28  >> [period, response] = Example03_05a(1, 100, 1, 0.2, time);
29  >> plot(time, response)
30  >> Example03_05a(1, 100, 1, 0.2, time)
31  ans =
32      0.6291
```


Help
MATLAB File Help: Example03_05a
MATLAB File Help: Example03_05a
View code for Example03_05a
Default Topics
Example03_05a
Under-damped free vibrations of a mass-spring-damper system
Input Arguments:
m = Mass, a scalar, SI unit: kg
k = Spring constant, a scalar, SI unit: N/m
c = Damping constant, a scalar, SI unit: N/(m/s)
delta = Initial displacement, a scalar, SI unit: m
t = Time, a row vector, SI unit: s
Output Arguments:
T = Period, a scalar, SI unit: s
x = Displacement, a row vector of the same length of t, SI unit: m
On Error, it prints a message and returns:
T = 0, a scalar
x = 0, a scalar

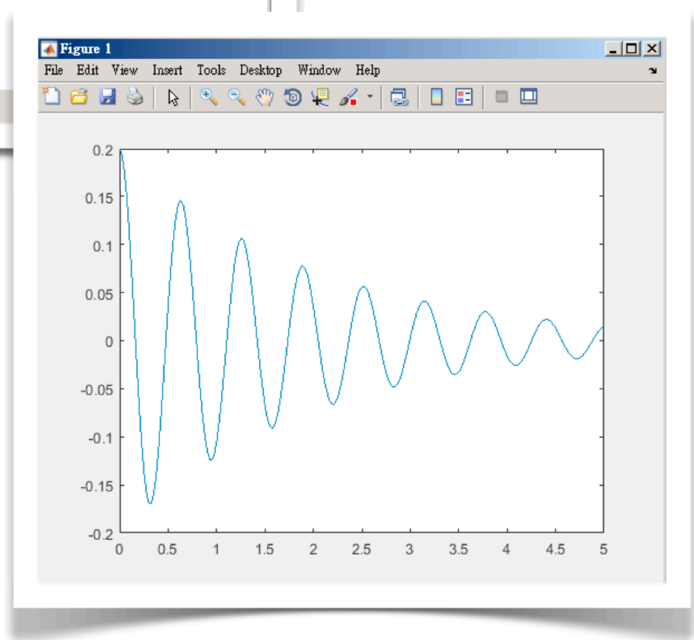


Figure 1
File Edit View Insert Tools Desktop Window Help
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

Example03_05c.m: Damped Free Vibrations

[9] Type, save, and run the following program, which is based on Example02_15b.m (page 119), demonstrating the calling of the function Example03_05a (lines 36 and 38). The output is the same as that in 2.15[8], page 119.

```
33  clear
34  mass = 1; spring = 100; damper = 1; delta = 0.2;
35  time = 0;
36  T = Example03_05a(mass, spring, damper, delta, time);
37  time = linspace(0, 3*T, 100);
38  [T, response] = Example03_05a(mass, spring, damper, delta, time);
39  axes('XTick', T:T:3*T, 'XTickLabel', {'T','2T','3T'});
40  axis([0, 3*T, -0.2, 0.2])
41  grid on
42  hold on
43  comet(time, response)
44  title('Damped Free Vibrations')
45  xlabel(['Time (T = ', num2str(T), ' sec)'])
46  ylabel('Displacement (m)')
```

3.6 Subfunctions

Example03_06a.m: Damped Free Vibrations

[2] In the following program, lines 1-14 are copied from Example03_05c.m (last page), and the function names in lines 4 and 6 are changed to UDFV (under-damped free vibrations) as shown. Lines 16-27 are copied from Example03_05a.m (page 134), and the function name in line 16 is changed to UDFV as shown. The output is the same as that in 2.15[8], page 119.

```
 1  clear
 2  mass = 1; spring = 100; damper = 1; delta = 0.2;
 3  time = 0;
 4  T = UDFV(mass, spring, damper, delta, time);
 5  time = linspace(0, 3*T, 100);
 6  [T, response] = UDFV(mass, spring, damper, delta, time);
 7  axes('XTick', T:T:3*T, 'XTickLabel', {'T','2T','3T'});
 8  axis([0, 3*T, -0.2, 0.2])
 9  grid on
10  hold on
11  comet(time, response)
12  title('Damped Free Vibrations')
13  xlabel(['Time (T = ', num2str(T), ' sec)'])
14  ylabel('Displacement (m)')
15
16  function [T, x] = UDFV(m, k, c, delta, t)
17  omega = sqrt(k/m);
18  cC = 2*m*omega;
19  if c>= cC
20      disp('Not an under-damped system!')
21      T = 0; x = 0;
22      return;
23  end
24  omegaD = omega*sqrt(1-(c/cC)^2);
25  T = 2*pi/omegaD;
26  x = delta*exp(-c*t/(2*m)).*(cos(omegaD*t)+c/(2*m*omegaD)*sin(omegaD*t));
27  end
```

Lines 1-14 are main program, while the function UDFV (lines 16-27) is a subfunction. Since the main program doesn't have any input arguments, it can be called by clicking the **Run** button (1.5[8], page 22). →

Example03_06b.m: MATLAB 2016a or Earlier Versions

[4] This program can be executed with MATLAB 2016a or earlier versions, but it cannot be opened as a **Live Script**. #

```
function Example03_06b
mass = 1; spring = 100; damper = 1; delta = 0.2;
time = 0;
T = UDFV(mass, spring, damper, delta, time);
time = linspace(0, 3*T, 100);
[T, response] = UDFV(mass, spring, damper, delta, time);
axes('XTick', T:T:3*T, 'XTickLabel', {'T','2T','3T'});
axis([0, 3*T, -0.2, 0.2])
grid on
hold on
comet(time, response)
title('Damped Free Vibrations')
xlabel(['Time (T = ', num2str(T), ' sec)'])
ylabel('Displacement (m)')
end

function [T, x] = UDFV(m, k, c, delta, t)
omega = sqrt(k/m);
cC = 2*m*omega;
if c>= cC
    disp('Not an under-damped system!')
    T = 0; x = 0;
    return;
end
omegaD = omega*sqrt(1-(c/cC)^2);
T = 2*pi/omegaD;
x = delta*exp(-c*t/(2*m)).*(cos(omegaD*t)+c/(2*m*omegaD)*sin(omegaD*t));
end
```

3.7 Nested Functions

Example03_07.m: Ball-Throwing

[1] Create a new file, copy all the lines in Example01_18.m (page 55), and make the following changes: insert lines 3 and 39 so that the main program becomes a function; indent all the lines of the function `pushbuttonCallback` (lines 29-38) as shown to emphasize that it is a **nested function** (this step is not really necessary); delete the `global`-statement (lines 2 and 29 in Example01_18.m) from both functions; add line 1 as a single-statement main program. Save as Example03_07.m, and run the program. The resulting GUI should be the same as that in 1.18[3-6], page 56. →

```
 1   Trajectory
 2
 3   function Trajectory
 4   g = 9.81;
 5   figure('Position', [30,70,500,400])
 6   axes('Units', 'pixels', ...
 7       'Position', [50,80,250,250])
 8   axis([0, 10, 0, 10])
 9   xlabel('Distance (m)'), ylabel('Height (m)')
10   title('Trajectory of a Ball')
11
12   uicontrol('Style', 'text', ...
13       'String', 'Initial velocity (m/s)', ...
14       'Position', [330,300,150,20])
15   velocityBox = uicontrol('Style', 'edit', ...
16       'String', '5', ...
17       'Position', [363,280,80,20]);
18   uicontrol('Style', 'text', ...
19       'String', 'Elevation angle (deg)', ...
20       'Position', [330,240,150,20])
21   angleBox = uicontrol('Style', 'edit', ...
22       'String', '45', ...
23       'Position', [363,220,80,20]);
24   uicontrol('Style', 'pushbutton', ...
25       'String', 'Throw', ...
26       'Position', [363,150,80,30], ...
27       'Callback', @pushbuttonCallback)
28
29       function pushbuttonCallback(pushButton, ~)
30           v0 = str2double(velocityBox.String);
31           theta = str2double(angleBox.String)*pi/180;
32           t1 = 2*v0*sin(theta)/g;
33           t = 0:0.01:t1;
34           x = v0*cos(theta)*t;
35           y = v0*sin(theta)*t-g*t.^2/2;
36           hold on
37           comet(x, y)
38       end
39   end
```

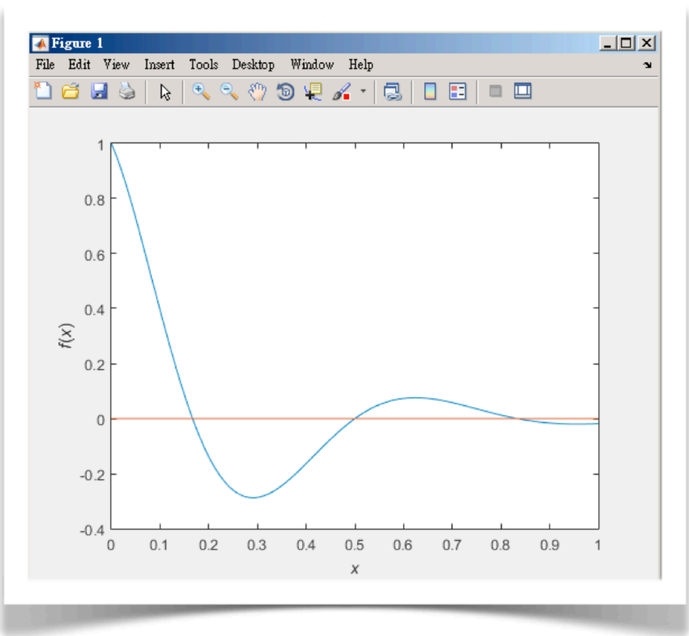
3.8 Function Handles

Example03_08a.m: Finding Zeros

[2] Create and run the following program, which finds zeros of the function $f(x) = e^{-4x}\cos 3\pi x$, i.e., solving the equation $e^{-4x}\cos 3\pi x = 0$.

```
clear
handle = @oscilation;
x1 = fzero(handle, 0.2)
x2 = fzero(handle, 0.5)
x3 = fzero(handle, 0.8)

function fx = oscillation(x)
fx = exp(-4*x)*cos(3*pi*x);
end
```

```
x1 =
    0.1667
x2 =
    0.5000
x3 =
    0.8333
```



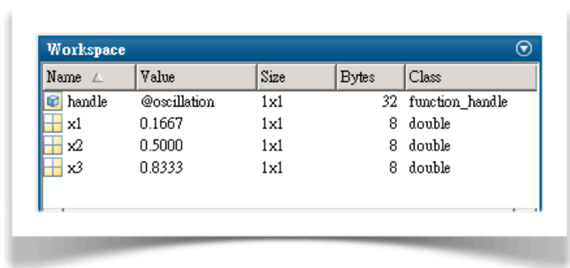


Table 3.8 Function Handles

Function	Description
`handle = @function`	Retrieve function handle
`handle(...)`	Evaluate function value

Details and More: Help>MATLAB>Language Fundamentals>Data Types>Function Handles

Example03_08b.m: Finding Zeros

[8] The following program implements a simplified version of the function `fzero`. Create a new file, copy all the lines in Example03_08a.m (page 141), make changes in lines 12 and 13 as shown, add a function `fzero` (lines 20-33), and run the program. The output is shown in [9].

```
10  clear
11  handle = @oscillation;
12  x1 = fzero(handle, 0.1)
13  x2 = fzero(handle, 0.4)
14  x3 = fzero(handle, 0.8)
15
16  function fx = oscillation(x)
17  fx = exp(-4*x)*cos(3*pi*x);
18  end
19
20  function x = fzero(handle, x0)
21  tolerance = 1.0e-6;
22  step = 0.01;
23  x = x0;
24  s1 = sign(handle(x));
25  while step/x > tolerance
26      if s1 == sign(handle(x+step))
27          x = x+step;
28      else
29          step = step/2;
30      end
31  end
32  disp('Simplified version')
33  end
```

```
Simplified version
x1 =
    0.1667
Simplified version
x2 =
    0.5000
Simplified version
x3 =
    0.8333
```

3.9 Anonymous Functions

Example03_09a.m: Anonymous Functions

[1] This script demonstrates the use of anonymous functions. It solves the nonlinear equation $e^{-4x}\cos 3\pi x = 0$ near 0.2, 0.5, and 0.8. It also evaluates the function values $f(x) = e^{-4x}\cos 3\pi x$ at 0.2, 0.5, and 0.8. The output is shown in [2].

```
clear
fun = @(x) exp(-4*x)*cos(3*pi*x);
x1 = fzero(fun, 0.2)
x2 = fzero(fun, 0.5)
x3 = fzero(fun, 0.8)
fx1 = fun(0.2)
fx2 = fun(0.5)
fx3 = fun(0.8)
```

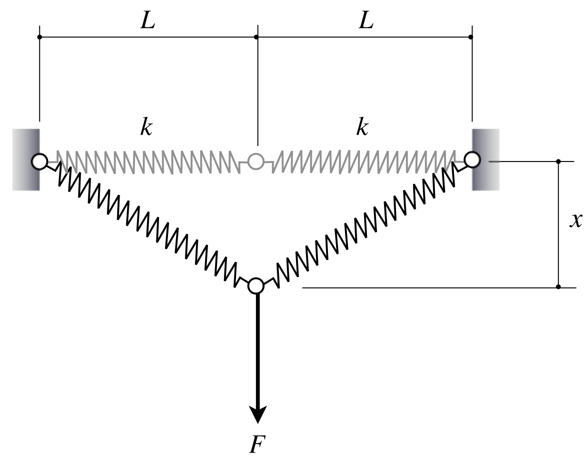
Note that, in this program, we use the built-in function `fzero`, not the user-defined function `fzero` in lines 20-33 of Example03_08b.m, last page.

```
x1 =
    0.1667
x2 =
    0.5000
x3 =
    0.8333
fx1 =
   -0.1389
fx2 =
  -2.4861e-17
fx3 =
    0.0126
```

Visibility of Variables

[4] An anonymous function can use previously defined variables. For example, Example03_09a.m could be rewritten as follows. →

```
clear, a = 4; b = 3;
fun = @(x) exp(-a*x)*cos(b*pi*x);
x1 = fzero(fun, 0.2)
x2 = fzero(fun, 0.5)
x3 = fzero(fun, 0.8)
fx1 = fun(0.2)
fx2 = fun(0.5)
fx3 = fun(0.8)
```



Example03_09b.m: Symmetrical Two-Spring System

[6] Create a new file, type following lines, and run the program.

```
clear
k = 6.8; L = 12; F = 9.2;
equation = @(x) 2*k*(sqrt(L^2+x^2)-L)*x/sqrt(L^2+x^2)-F;
fzero(equation, 0)
```

Note that, in this program, we use the built-in function `fzero`, not the user-defined function `fzero` in lines 20-33 of Example03_08b.m, page 143.

```
x =
    6.1485
```

3.10 Function Precedence Order

Example03_10.m

[2] On the **Command Window**, execute the following commands. The output is shown in [3].

```
clear
a = sin(1)
sin = 3:5;
b = sin(1)
clear
c = sin(1)
```

```
>> clear
>> a = sin(1)
a =
     0.8415
>> sin = 3:5;
>> b = sin(1)
b =
       3
>> clear
>> c = sin(1)
c =
     0.8415
```

Table 3.10 Function Precedence Order

Description	Precedence level
Variables	1
Nested functions	3
Subfunction (local functions)	4
Functions in current folder	9
Functions in the search path	10

Details and More:
Help>MATLAB>Programming Scripts and Functions>Functions>Function Basics>Function Precedence Order

3.11 Program Files

Example03_11.m: Damped Free Vibrations

[2] This program is a modified version of Example03_05c.m. This program file (Example03_11.m) and the program file Example03_05a.m constitute a program. Run the program, using the following input data (see [3], next page): `m` = 1 kg, `k` = 100 N/m, `c` = 1 N/(m/s), and `delta` = 0.2 m. The graphic output is shown in 2.15[8], page 119. →

```
clear
[mass, spring, damper, delta] = askProperties
time = 0;
T = Example03_05a(mass, spring, damper, delta, time);
time = linspace(0, 3*T, 100);
[T, response] = Example03_05a(mass, spring, damper, delta, time);
plotDisplacement(T, time, response)

function [m, k, c, delta] = askProperties
m = input('Enter the mass (kg): ');
k = input('Enter the spring constant (N/m): ');
c = input('Enter the damper constant (N/(m/s)): ');
delta = input('Enter the initial displacement (m): ');
end

function plotDisplacement(T, time, response)
axes('XTick', T:T:3*T, 'XTickLabel', {'T','2T','3T'});
axis([0, 3*T, -0.2, 0.2])
grid on
hold on
comet(time, response)
title('Damped Free Vibrations')
xlabel(['Time (T = ', num2str(T), ' sec)'])
ylabel('Displacement (m)')
end
```

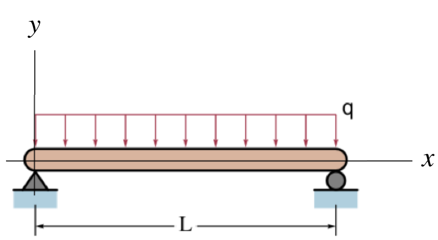
```
>> Example03_11
Enter the mass (kg): 1
Enter the spring constant (N/m): 100
Enter the damper constant (N/(m/s)): 1
Enter the initial displacement (m): 0.2
mass =
     1
spring =
   100
damper =
     1
delta =
    0.2000
>>
```

3.12 Example: Deflection of Beams

Example03_12.m: Deflection of Beams

[1] The following program calculates the deflection at the center of a simply supported beam subject to a uniform load q (SI unit: N/m) as shown in [2] (*Figure source: https://commons.wikimedia.org/wiki/File:Simple_beam_with_uniform_distributed_load.svg, by Hermanoere*). It uses the formulas in 2.14[1] (page 115). The uniform load q can be thought of as n evenly spaced concentrated forces $F = qL/n$, with a spacing of L/n. If n is large enough (e.g., $n = 1000$), we'll obtain a solution close to the theoretical solution. We assume $q = 500$ N/m and the same properties for the beam as those in 2.14, $w = 0.1$ m, $h = 0.1$ m, $L = 8$ m, $E = 210$ GPa.

```
w = 0.1; h = 0.1; L = 8; E = 210e9; q = 500;
n = 1000; F = q*L/n;
delta = 0;
for k = 1:n
    a = (L/n)*k;
    delta = delta + deflection(w, h, L, E, F, a, L/2);
end
fprintf('Deflection at center is %.4f mm\n', delta*1000)

function delta = deflection(w, h, L, E, F, a, x)
I = w*h^3/12;
R = F/L*(L-a);
theta = F*a/(6*E*I*L)*(2*L-a)*(L-a);
delta = theta*x-R*x.^3/(6*E*I)+F/(6*E*I)*((x>a).*((x-a).^3));
end
```




```
Deflection at center is 15.2381 mm
```

Verification of the Result

[5] The result [3], last page, can be verified using a well-know formula (see *Wikipedia>Deflection (engineering)*):

$$\delta = \frac{5qL^4}{384EI}$$

which can be easily calculated with the following commands:

```
>> w = 0.1; h = 0.1; L = 8;
>> E = 210e9; q = 500;
>> I = w*h^3/12;
>> delta = 5*q*L^4/(384*E*I)*1000
delta =
   15.2381
```

Arbitrary Loads

This section demonstrates the calculation of deflection for a particular case of loads. The same idea can be used to calculate the deflection of a simply supported beam subject to any concentrated and/or distributed loads. #

3.13 Example: Sorting and Searching

Example03_13.m: Sorting and Searching

[2] This program performs sorting and searching of numbers. Before looking into the statements, test-run the program as shown in [4], next page. (Continued at [3], next page.) →

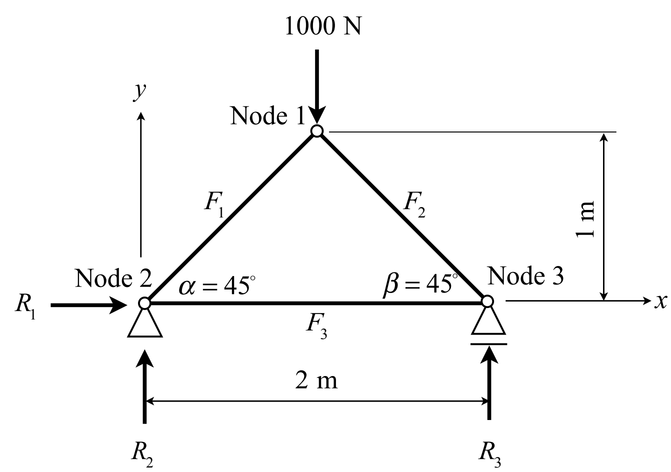
```
a = []; n = 0;
disp('1. Input numbers and sort')
disp('2. Display the list')
disp('3. Search')
disp('4. Save')
disp('5. Load')
disp('6. Quit')
while 1
    task = input('Enter a task number: ');
    switch task
        case 1
            while 1
                string = input('Enter a number (or stop): ', 's');
                if strcmpi(string, 'stop')
                    break;
                else
                    n = n+1;
                    a(n) = str2num(string);
                end
            end
            a = sort(a);
        case 2
            disp(a)
        case 3
            key = input('Enter a key number: ');
            found = search(a, key);
            if found
                disp(['Index = ', num2str(found)])
            else
                disp('Not found!')
            end
        case 4
            save('Datafile03_13', 'a')
        case 5
            load('Datafile03_13')
            n = length(a);
        case 6
            break
    end
end

```

[3] Example03_13.m (Continued)

```
function out = sort(a)
n = length(a);
for i = n-1:-1:1
    for j = 1:i
        if a(j) > a(j+1)
            tmp = a(j);
            a(j) = a(j+1);
            a(j+1) = tmp;
        end
    end
end
out = a;
end

function found = search(a, key)
n = length(a);
low = 1;
high = n;
found = 0;
while low <= high && ~found
    mid = floor((low+high)/2);
    if key == a(mid)
        found = mid;
    elseif key < a(mid)
        high = mid-1;
    else
        low = mid+1;
    end
end
end
```

```
>> Example03_13
1. Input numbers and sort
2. Display the list
3. Search
4. Save
5. Load
6. Quit
Enter a task number: 1
Enter a number (or stop): 6
Enter a number (or stop): 9
Enter a number (or stop): 3
Enter a number (or stop): 5
Enter a number (or stop): 1
Enter a number (or stop): 7
Enter a number (or stop): stop
Enter a task number: 2
     1     3     5     6     7     9
Enter a task number: 3
Enter a key number: 6
Index = 4
Enter a task number: 4
Enter a task number: 6
>> Example03_13
1. Input numbers and sort
2. Display the list
3. Search
4. Save
5. Load
6. Quit
Enter a task number: 5
Enter a task number: 2
     1     3     5     6     7     9
Enter a task number: 6
>>
```

3.14 Example: Statically Determinate Trusses (Version 1.0)



[2] The system of equations in Eq. (b) can be easily solved with the following MATLAB commands:

```
>> a = sqrt(2)/2;
>> A = [-a  a  0  0  0  0;
        -a -a  0  0  0  0;
         a  0  1  1  0  0;
         a  0  0  0  1  0;
         0 -a -1  0  0  0;
         0  a  0  0  0  1];
>> b = [0, 1000, 0, 0, 0, 0]';
>> x = A\b
x =
 -707.1068
 -707.1068
  500.0000
         0
  500.0000
  500.0000
```

We have $F_1 = F_2 = -707.1068$ N, $F_3 = 500$ N, $R_1 = 0$, and $R_2 = R_3 = 500$ N.

The last command (`x = A\b`) solves a system of linear equations `A*x = b`. Note that `b` is a column vector. The matrices `A` and `b` have the same number of rows. We'll give you more details about this **left divide operator** (or **backslash operator**) in Sections 9.7 and 10.3.

Example03_14.m: Truss 1.0

[4] This program solves the 3-bar truss problem in [1], last page. (Continued at [5], next page.) →

```
clear, Nodes = [
    1, 1, 0, 0, 0, -1000, 0,   0;
    0, 0, 1, 1, 0,     0, 0, 500;
    2, 0, 0, 1, 0,     0, 0, 500];
Members = [1, 2; 1, 3; 2, 3];
[Nodes, Members] = solveTruss(Nodes, Members)

```

[5] Example03_14.m (Continued). The output is shown in [6-7], next page. →

```
function [outNodes, outMembers] = solveTruss(Nodes, Members)
n = size(Nodes,1); m = size(Members,1);
if (m+3) < 2*n
    disp('Unstable!')
    outNodes = 0; outMembers = 0; return
elseif (m+3) > 2*n
    disp('Statically indeterminate!')
    outNodes = 0; outMembers = 0; return
end
A = zeros(2*n, 2*n); loads = zeros(2*n,1); nsupport = 0;
for i = 1:n
    for j = 1:m
        if Members(j,1) == i || Members(j,2) == i
            if Members(j,1) == i
                n1 = i; n2 = Members(j,2);
            elseif Members(j,2) == i
                n1 = i; n2 = Members(j,1);
            end
            x1 = Nodes(n1,1); y1 = Nodes(n1,2);
            x2 = Nodes(n2,1); y2 = Nodes(n2,2);
            L = sqrt((x2-x1)^2+(y2-y1)^2);
            A(2*i-1,j) = (x2-x1)/L;
            A(2*i,  j) = (y2-y1)/L;
        end
    end
    if (Nodes(i,3) == 1)
        nsupport = nsupport+1;
        A(2*i-1,m+nsupport) = 1;
    end
    if (Nodes(i,4) == 1)
        nsupport = nsupport+1;
        A(2*i, m+nsupport) = 1;
    end
    loads(2*i-1) = -Nodes(i,5);
    loads(2*i)   = -Nodes(i,6);
end
forces = A\loads;
Members(:,3) = forces(1:m);
nsupport = 0;
for i = 1:n
    if (Nodes(i,3) == 1)
        nsupport = nsupport+1;
        Nodes(i,7) = forces(m+nsupport);
    end
    if (Nodes(i,4) == 1)
        nsupport = nsupport+1;
        Nodes(i,8) = forces(m+nsupport);
    end
end
outNodes = Nodes; outMembers = Members;
disp('Solved successfully.')
end
```

```
>> Example03_14
Solved successfully.
Nodes =
         1         1         0         0         0     -1000         0         0
         0         0         1         1         0         0         0       500
         2         0         0         1         0         0         0       500
Members =
    1.0000    2.0000 -707.1068
    1.0000    3.0000 -707.1068
    2.0000    3.0000  500.0000
>>
```

Table 3.14a Nodal Data for 3-Bar Truss

node	*x*	*y*	*supportx*	*supporty*	*loadx*	*loady*	*reactionx*	*reactiony*
1	1	1	0	0	0	-1000	0	0
2	0	0	1	1	0	0	0	500
3	2	0	0	1	0	0	0	500

Table 3.14b Member Data for 3-Bar Truss

member	*node1*	*node2*	*force*
1	1	2	-707.11
2	1	3	-707.11
3	2	3	500

3.15 Example: Statically Determinate Trusses (Version 2.0)

Example03_15.m: Truss 2.0

[1] This is an improved version of Example03_14.m. It implements a text-based user-interface, allowing the user to define a statically determinate truss. We'll use this program to solve two truss problems: the 3-bar truss problem (page 154) is solved in [6] (page 164), and a 21-bar truss problem ([7], page 165) is solved in [8-11] (pages 165-167).

```
clear
Nodes = []; Members = [];
disp(' 1. Input nodal coordinates')
disp(' 2. Input connecting nodes of members')
disp(' 3. Input three supports')
disp(' 4. Input loads')
disp(' 5. Print truss')
disp(' 6. Solve truss')
disp(' 7. Print results')
disp(' 8. Save data')
disp(' 9. Load data')
disp('10. Quit')
while 1
    task = input('Enter the task number: ');
    switch task
        case 1
            Nodes = inputNodes(Nodes);
        case 2
            Members = inputMembers(Members);
        case 3
            Nodes = inputSupports(Nodes);
        case 4
            Nodes = inputLoads(Nodes);
        case 5
            printTruss(Nodes, Members)
        case 6
            [Nodes, Members] = solveTruss(Nodes, Members);
        case 7
            printResults(Nodes, Members)
        case 8
            saveAll(Nodes, Members)
        case 9
            [Nodes, Members] = loadAll;
        case 10
            break
    end
end
```

(Continued at [2] on the next page.) →

[2] Example03_15.m (Continued)

```
function output = inputNodes(Nodes)
while 1
    data = input('Enter [node, x, y] or 0 to stop: ');
    if data(1) == 0
        break
    else
        Nodes(data(1),1:2) = data(2:3);
    end
end
output = Nodes;
end

function output = inputMembers(Members)
m = 0;
while 1
    data = input('Enter [node1, node2] or 0 to stop: ');
    if data(1) == 0
        break
    else
        m = m+1;
        Members(m,1:2) = data;
    end
end
output = Members;
end

function output = inputSupports(Nodes)
Nodes(:,3:4) = 0;
for k = 1:3
    data = input('Enter [node, dir] (dir: ''x'' or ''y''): ');
    if data(2) == 'x'
        Nodes(data(1),3) = 1;
    elseif data(2) == 'y'
        Nodes(data(1),4) = 1;
    end
end
output = Nodes;
end

function output = inputLoads(Nodes)
Nodes(:,5:6) = 0;
while 1
    data = input('Enter [node, load-x, load-y] or 0 to stop: ');
    if data(1) == 0
        break
    else
        Nodes(data(1),5:6) = data(2:3);
    end
end
output = Nodes;
end
```

(Continued at [3], next page.) →

[3] Example03_15.m (Continued)

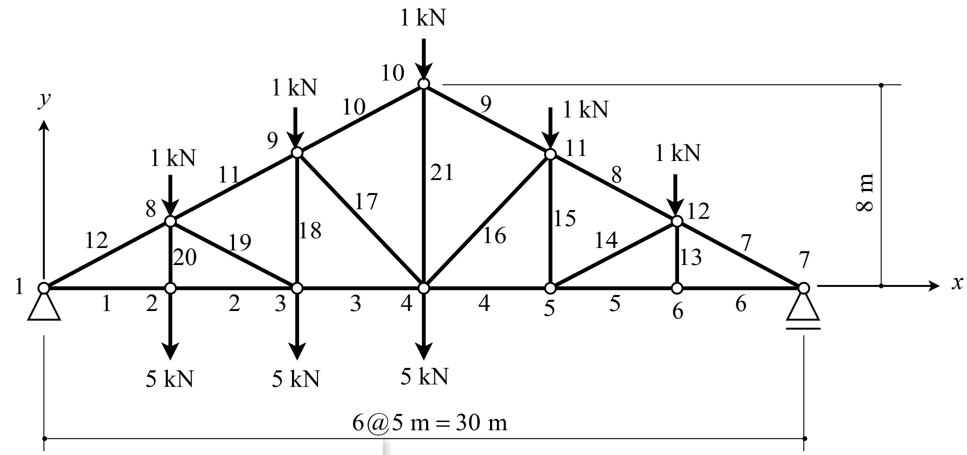
```
function printTruss(Nodes, Members)
if (size(Nodes,2)<6 || size(Members,2)<2)
    disp('Truss data not complete'); return
end
fprintf('\nNodal Data\n')
fprintf('Node       x        y  Support-x  Support-y   Load-x   Load-y\n')
for k = 1:size(Nodes,1)
    fprintf('%4.0f%9.2f%9.2f%11.0f%11.0f%9.0f%9.0f\n', k, Nodes(k, 1:6))
end
fprintf('\nMember Data\n')
fprintf('Member  Node1    Node2\n')
for k = 1:size(Members,1)
    fprintf('%4.0f%9.0f%9.0f\n', k, Members(k, 1:2))
end
end

function printResults(Nodes, Members)
if (size(Nodes,2)<8 || size(Members,2)<3)
    disp('Results not available!'), return
end
fprintf('\nReaction Forces\n')
fprintf('Node  Reaction-x  Reaction-y\n')
for k = 1:size(Nodes,1)
    fprintf('%4.0f%12.2f%12.2f\n', k, Nodes(k, 7:8))
end
fprintf('\nMember Forces\n')
fprintf('Member     Force\n')
for k = 1:size(Members,1)
    fprintf('%4.0f%12.2f\n', k, Members(k, 3))
end
end

function saveAll(Nodes, Members)
fileName = input('Enter file name (default Datafile): ', 's');
if isempty(fileName)
    fileName = 'Datafile';
end
save(fileName, 'Nodes', 'Members')
end

function [Nodes, Members] = loadAll
fileName = input('Enter file name (default Datafile): ', 's');
if isempty(fileName)
    fileName = 'Datafile';
end
load(fileName)
end

```

(Continued at [4], next page.) →

[4] Example03_15.m (Continued) →

```
function [outNodes, outMembers] = solveTruss(Nodes, Members)
n = size(Nodes,1); m = size(Members,1);
if (m+3) < 2*n
    disp('Unstable!')
    outNodes = 0; outMembers = 0; return
elseif (m+3) > 2*n
    disp('Statically indeterminate!')
    outNodes = 0; outMembers = 0; return
end
A = zeros(2*n, 2*n); loads = zeros(2*n,1); nsupport = 0;
for i = 1:n
    for j = 1:m
        if Members(j,1) == i || Members(j,2) == i
            if Members(j,1) == i
                n1 = i; n2 = Members(j,2);
            elseif Members(j,2) == i
                n1 = i; n2 = Members(j,1);
            end
            x1 = Nodes(n1,1); y1 = Nodes(n1,2);
            x2 = Nodes(n2,1); y2 = Nodes(n2,2);
            L = sqrt((x2-x1)^2+(y2-y1)^2);
            A(2*i-1,j) = (x2-x1)/L;
            A(2*i,  j) = (y2-y1)/L;
        end
    end
    if (Nodes(i,3) == 1)
        nsupport = nsupport+1;
        A(2*i-1,m+nsupport) = 1;
    end
    if (Nodes(i,4) == 1)
        nsupport = nsupport+1;
        A(2*i, m+nsupport) = 1;
    end
    loads(2*i-1) = -Nodes(i,5);
    loads(2*i)   = -Nodes(i,6);
end
forces = A\loads;
Members(:,3) = forces(1:m);
nsupport = 0;
for i = 1:n
    if (Nodes(i,3) == 1)
        nsupport = nsupport+1;
        Nodes(i,7) = forces(m+nsupport);
    end
    if (Nodes(i,4) == 1)
        nsupport = nsupport+1;
        Nodes(i,8) = forces(m+nsupport);
    end
end
outNodes = Nodes; outMembers = Members;
disp('Solved successfully.')
end
```

```
>> Example03_15
 1. Input nodal coordinates
 2. Input connecting nodes of members
 3. Input three supports
 4. Input loads
 5. Print truss
 6. Solve truss
 7. Print results
 8. Save data
 9. Load data
10. Quit
Enter the task number: 1
Enter [node, x, y] or 0 to stop: [1 1 1]
Enter [node, x, y] or 0 to stop: [2 0 0]
Enter [node, x, y] or 0 to stop: [3 2 0]
Enter [node, x, y] or 0 to stop: 0
Enter the task number: 2
Enter [node1, node2] or 0 to stop: [1 2]
Enter [node1, node2] or 0 to stop: [1 3]
Enter [node1, node2] or 0 to stop: [2 3]
Enter [node1, node2] or 0 to stop: 0
Enter the task number: 3
Enter [node, dir] (dir: 'x' or 'y'): [2 'x']
Enter [node, dir] (dir: 'x' or 'y'): [2 'y']
Enter [node, dir] (dir: 'x' or 'y'): [3 'y']
Enter the task number: 4
Enter [node, load-x, load-y] or 0 to stop: [1 0 -1000]
Enter [node, load-x, load-y] or 0 to stop: 0
Enter the task number: 5

Nodal Data
Node        x        y  Support-x  Support-y   Load-x   Load-y
   1     1.00     1.00          0          0        0    -1000
   2     0.00     0.00          1          1        0        0
   3     2.00     0.00          0          1        0        0

Member Data
Member  Node1    Node2
   1        1        2
   2        1        3
   3        2        3
Enter the task number: 6
Solved successfully.
Enter the task number: 7

Reaction Forces
Node  Reaction-x  Reaction-y
   1        0.00        0.00
   2        0.00      500.00
   3        0.00      500.00

Member Forces
Member     Force
   1     -707.11
   2     -707.11
   3      500.00
Enter the task number: 8
Enter file name (default Datafile): Datafile03_15a
Enter the task number: 10
>>
```




```
>> Example03_15
 1. Input nodal coordinates
 2. Input connecting nodes of members
 3. Input three supports
 4. Input loads
 5. Print truss
 6. Solve truss
 7. Print results
 8. Save data
 9. Load data
10. Quit
Enter the task number: 1
Enter [node, x, y] or 0 to stop: [1 0 0]
Enter [node, x, y] or 0 to stop: [2 5 0]
Enter [node, x, y] or 0 to stop: [3 10 0]
Enter [node, x, y] or 0 to stop: [4 15 0]
Enter [node, x, y] or 0 to stop: [5 20 0]
Enter [node, x, y] or 0 to stop: [6 25 0]
Enter [node, x, y] or 0 to stop: [7 30 0]
Enter [node, x, y] or 0 to stop: [8 5 8/3]
Enter [node, x, y] or 0 to stop: [9 10 8*2/3]
Enter [node, x, y] or 0 to stop: [10 15 8]
Enter [node, x, y] or 0 to stop: [11 20 8*2/3]
Enter [node, x, y] or 0 to stop: [12 25 8/3]
Enter [node, x, y] or 0 to stop: 0
Enter the task number: 2
Enter [node1, node2] or 0 to stop: [1 2]
Enter [node1, node2] or 0 to stop: [2 3]
Enter [node1, node2] or 0 to stop: [3 4]
Enter [node1, node2] or 0 to stop: [4 5]
Enter [node1, node2] or 0 to stop: [5 6]
Enter [node1, node2] or 0 to stop: [6 7]
Enter [node1, node2] or 0 to stop: [7 12]
Enter [node1, node2] or 0 to stop: [12 11]
Enter [node1, node2] or 0 to stop: [11 10]
Enter [node1, node2] or 0 to stop: [10 9]
Enter [node1, node2] or 0 to stop: [9 8]
Enter [node1, node2] or 0 to stop: [8 1]
Enter [node1, node2] or 0 to stop: [12 6]
Enter [node1, node2] or 0 to stop: [12 5]
Enter [node1, node2] or 0 to stop: [11 5]
Enter [node1, node2] or 0 to stop: [11 4]
Enter [node1, node2] or 0 to stop: [9 4]
Enter [node1, node2] or 0 to stop: [9 3]
Enter [node1, node2] or 0 to stop: [8 3]
Enter [node1, node2] or 0 to stop: [8 2]
Enter [node1, node2] or 0 to stop: [10 4]
Enter [node1, node2] or 0 to stop: 0
```

```
Enter the task number: 3
Enter [node, dir] (dir: 'x' or 'y'): [1 'x']
Enter [node, dir] (dir: 'x' or 'y'): [1 'y']
Enter [node, dir] (dir: 'x' or 'y'): [7 'y']
Enter the task number: 4
Enter [node, load-x, load-y] or 0 to stop: [2 0 -5000]
Enter [node, load-x, load-y] or 0 to stop: [3 0 -5000]
Enter [node, load-x, load-y] or 0 to stop: [4 0 -5000]
Enter [node, load-x, load-y] or 0 to stop: [8 0 -1000]
Enter [node, load-x, load-y] or 0 to stop: [9 0 -1000]
Enter [node, load-x, load-y] or 0 to stop: [10 0 -1000]
Enter [node, load-x, load-y] or 0 to stop: [11 0 -1000]
Enter [node, load-x, load-y] or 0 to stop: [12 0 -1000]
Enter [node, load-x, load-y] or 0 to stop: 0
Enter the task number: 5

Nodal Data
Node         x         y  Support-x  Support-y   Load-x   Load-y
   1      0.00      0.00          1          1        0        0
   2      5.00      0.00          0          0        0    -5000
   3     10.00      0.00          0          0        0    -5000
   4     15.00      0.00          0          0        0    -5000
   5     20.00      0.00          0          0        0        0
   6     25.00      0.00          0          0        0        0
   7     30.00      0.00          0          1        0        0
   8      5.00      2.67          0          0        0    -1000
   9     10.00      5.33          0          0        0    -1000
  10     15.00      8.00          0          0        0    -1000
  11     20.00      5.33          0          0        0    -1000
  12     25.00      2.67          0          0        0    -1000

Member Data
Member  Node1    Node2
   1        1        2
   2        2        3
   3        3        4
   4        4        5
   5        5        6
   6        6        7
   7        7       12
   8       12       11
   9       11       10
  10       10        9
  11        9        8
  12        8        1
  13       12        6
  14       12        5
  15       11        5
  16       11        4
  17        9        4
  18        9        3
  19        8        3
  20        8        2
  21       10        4
Enter the task number: 6
Solved successfully.
```

```
Enter the task number: 7

Reaction Forces
Node  Reaction-x  Reaction-y
   1        0.00    12500.00
   2        0.00        0.00
   3        0.00        0.00
   4        0.00        0.00
   5        0.00        0.00
   6        0.00        0.00
   7        0.00     7500.00
   8        0.00        0.00
   9        0.00        0.00
  10        0.00        0.00
  11        0.00        0.00
  12        0.00        0.00

Member Forces
Member      Force
   1     23437.50
   2     23437.50
   3     17812.50
   4     13125.00
   5     14062.50
   6     14062.50
   7    -15937.50
   8    -14875.00
   9    -13812.50
  10    -13812.50
  11    -20187.50
  12    -26562.50
  13         0.00
  14     -1062.50
  15       500.00
  16     -1370.73
  17     -8224.39
  18      8000.00
  19     -6375.00
  20      5000.00
  21     12000.00
Enter the task number: 8
Enter file name (default Datafile): Datafile03_15b
Enter the task number: 10
>>
```

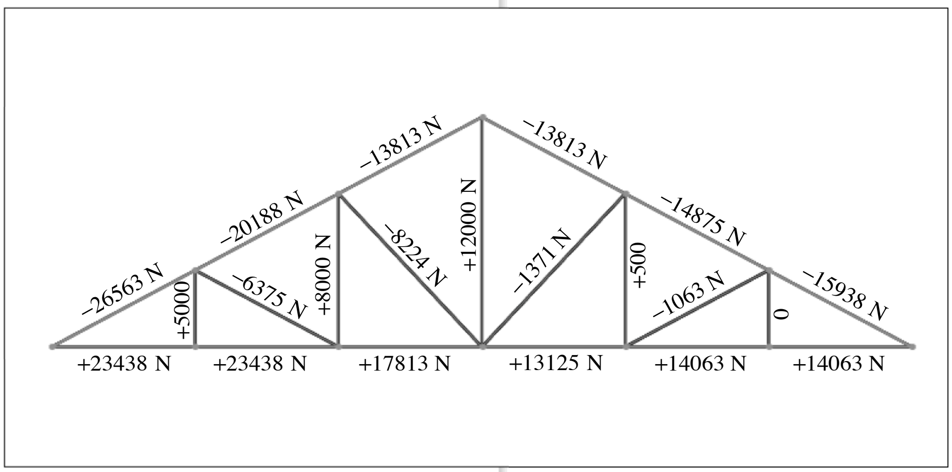


Table 3.15a Nodal Data for the 21-Bar Truss

node	*x*	*y*	*supportx*	*supporty*	*loadx*	*loady*	*reactionx*	*reactiony*
1	0	0	1	1	0	0	0	12500
2	5	0	0	0	0	-5000	0	0
3	10	0	0	0	0	-5000	0	0
4	15	0	0	0	0	-5000	0	0
5	20	0	0	0	0	0	0	0
6	25	0	0	0	0	0	0	0
7	30	0	0	1	0	0	0	7500
8	5	2.6667	0	0	0	-1000	0	0
9	10	5.3333	0	0	0	-1000	0	0
10	15	8	0	0	0	-1000	0	0
11	20	5.3333	0	0	0	-1000	0	0
12	25	2.6667	0	0	0	-1000	0	0

Table 3.15b Member Data for the 21-Bar Truss

member	*node1*	*node2*	*force*
1	1	2	23437.5
2	2	3	23437.5
3	3	4	17812.5
4	4	5	13125
5	5	6	14062.5
6	6	7	14062.5
7	7	12	-15937.5
8	12	11	-14875
9	11	10	-13812.5
10	10	9	-13812.5
11	9	8	-20187.5
12	8	1	-26562.5
13	12	6	0
14	12	5	-1062.5
15	11	5	500
16	11	4	-1370.73
17	9	4	-8224.39
18	9	3	8000
19	8	3	-6375
20	8	2	5000
21	10	4	12000

3.16 Additional Exercise Problems

Problem03_03: Functions

We mentioned, in 3.5[10] (page 136), that the purpose of using functions is modularization and abstraction. In this exercise, we use a short program to illustrate the ideas. This program is so short that you may not be able to appreciate the ideas of modularization and abstraction. However, as a program becomes large, these ideas are useful.

Consider Problem02_03 (page 122) again, and generate the same graphic output as before. This time, organize the program into a main function and three subfunctions. This is the main function:

```
clear
[n,p] = getData;
x = 0:n;
fx = Binomial(p, n, x);
drawCurve(x, fx, p, n)
end
```

You are asked to implement the three subfunctions: `getData`, `Binomial`, and `drawCurve`. Function `getData` allows the user to input the values of `n` and `p`. Function `Binomial` calculates $f(x)$ of the binomial distribution (see Problem02_03, page 122). Function `drawCurve` produces the graphic output.

Chapter 4

Cell Arrays, Structures, Tables, and User-Defined Classes

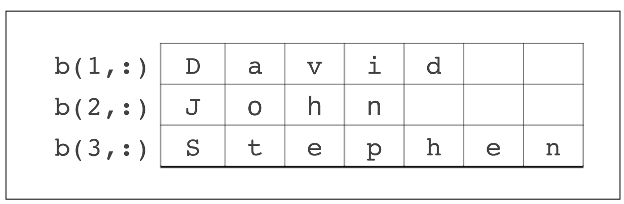
A data type is also called a **class**. The built-in classes we've discussed in Chapter 3 include numeric classes (**double**, **int32**, etc), **character**, **logical**, and **function handle**. **Cell array** and **structure** mentioned in Chapter 1 are also built-in classes. A class is defined by a set of lower-level **data** and **operations** on the data. For example, the class **double** has a binary data representation shown in 2.3[1] (page 74) and its operations include plus, minus, times, etc. It is possible to create **user-defined classes**. In a user-defined class, the data are called the **properties**, and the operations are called the **methods**.

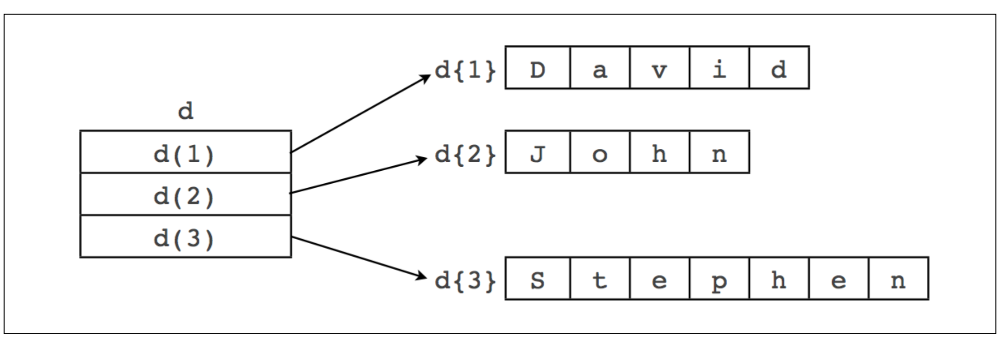
4.1 Cell Arrays

Example04_01a.m

[2] These commands demonstrate some important concepts about ordinary arrays and cell arrays. A screen output is shown in [5], next page. Schematic diagrams of the array `b` (or `c`) and `d` are shown in [3-4]; they are further explained in [6], next page.

```
clear
a = ['David', 'John', 'Stephen']
b = ['David  '; 'John   '; 'Stephen']
size(b)
c = char('David', 'John', 'Stephen')
size(c)
d = {'David', 'John', 'Stephen'}
d(1)
d{1}
d{1}(1)
d{3}(7)
```





```
>> clear
>> a = ['David', 'John', 'Stephen']
a =
    'DavidJohnStephen'
>> b = ['David  '; 'John   '; 'Stephen']
b =
  3×7 char array
    'David  '
    'John   '
    'Stephen'
>> size(b)
ans =
     3     7
>> c = char('David', 'John', 'Stephen')
c =
  3×7 char array
    'David  '
    'John   '
    'Stephen'
>> size(c)
ans =
     3     7
>> d = {'David', 'John', 'Stephen'}
d =
  1×3 cell array
    'David'    'John'    'Stephen'
>> d(1)
ans =
  cell
    'David'
>> d{1}
ans =
    'David'
>> d{1}(1)
ans =
    'D'
>> d{3}(7)
ans =
    'n'
>>
```

Example04_01b.m

[7] Like an ordinary array, a cell array may be multi-dimensional, as demonstrated in the following commands. The text output is shown in [8] and the graphic output is in [9].

```
clear
a = 45;
b = 'David';
c = [1, 2, 3];
d = [4, 5; 6, 7];
e = {a, b; c, d}
disp(e)
celldisp(e)
cellplot(e)
e{1,1}
e{2,1}(2)
e{1,2}(1)
e{2,2}(2,1)
```

```
>> clear
>> a = 45;
>> b = 'David';
>> c = [1, 2, 3];
>> d = [4, 5; 6, 7];
>> e = {a, b; c, d}
e =
  2×2 cell array
    [          45]    'David'
    [1×3 double]    [2×2 double]
>> disp(e)
    [          45]    'David'
    [1×3 double]    [2×2 double]
>> celldisp(e)
e{1,1} =
    45
e{2,1} =
     1     2     3
e{1,2} =
David
e{2,2} =
     4     5
     6     7
>> cellplot(e)
>> e{1,1}
ans =
    45
>> e{2,1}(2)
ans =
     2
>> e{1,2}(1)
ans =
    'D'
>> e{2,2}(2,1)
ans =
     6
>>
```

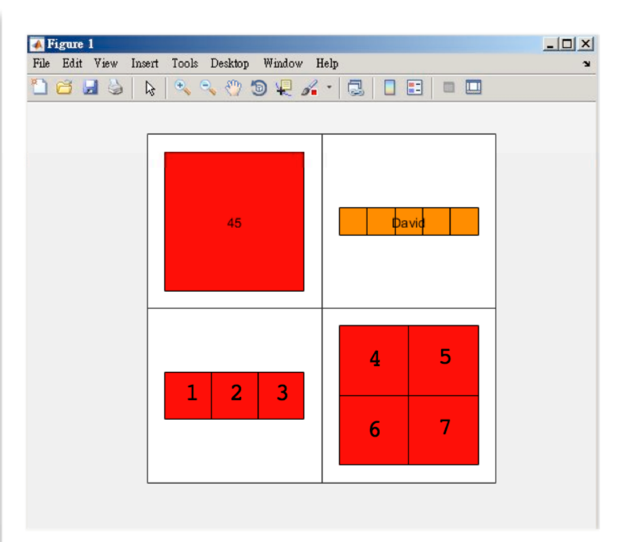


Table 4.1 Cell Arrays

Function	Description
`c = cell(n,m)`	Create an n-by-m cell array
`celldisp(c)`	Display cell array contents
`cellplot(c)`	Graphically display cell array
`c = struct2cell(s)`	Convert structure to cell array
`c = table2cell(t)`	Convert table to cell array
`s = cell2struct(c)`	Convert cell array to structure
`t = cell2table(c)`	Convert table to structure

Details and More: Help>MATLAB>Language Fundamentals>Data Types>Cell Arrays

4.2 Functions of Variable-Length Arguments

Example04_02a.m: Variable-Length Input Arguments

[1] Using cell arrays, a function may have variable-length arguments. The following program includes a function (lines 8-40) that allows variable-length input arguments. The function `area` (lines 8-40) calculates the area of a circle, square, rectangle, or triangle, according to its first input argument. Lines 2-6 are examples of using this function; their output are shown in [2], next page. →

```
 1  clear
 2  p = area('circle', 5)
 3  q = area('square', 6)
 4  s = area('rectangle', 7, 8)
 5  t = area('triangle', 'bh', 9, 10)
 6  u = area('triangle', 'abc', 5, 6, 7)
 7
 8  function output = area(varargin)
 9  output = 0;
10  switch varargin{1}
11      case 'circle'
12          if nargin == 2
13              r = varargin{2};
14              output = pi*r^2;
15          end
16      case 'square'
17          if nargin == 2
18              a = varargin{2};
19              output = a^2;
20          end
21      case 'rectangle'
22          if nargin == 3
23              b = varargin{2};
24              h = varargin{3};
25              output = b*h;
26          end
27      case 'triangle'
28          if strcmp(varargin{2},'bh') && nargin == 4
29              b = varargin{3};
30              h = varargin{4};
31              output = b*h/2;
32          elseif strcmp(varargin{2}, 'abc') && nargin == 5
33              a = varargin{3};
34              b = varargin{4};
35              c = varargin{5};
36              R = a*b*c/sqrt((a+b+c)*(a-b+c)*(b-c+a)*(c-a+b));
37              output = a*b*c/(4*R);
38          end
39  end
40  end
```

```
p =
   78.5398
q =
    36
s =
    56
t =
    45
u =
   14.6969
```

Example04_02b.m: Variable-Length Input/Output

[4] The following program includes a function (lines 49-78) that not only allows variable-length input arguments but also allows variable-length output arguments. The function `properties` (lines 49-78), in addition to calculate the area of a shape, can also calculate the moment of inertia of the shape (*Wikipedia>Second moment of area*), according to the number of output arguments provided by the calling function. See [5], next page, for an output. →

```
41  clear
42  Ac = properties('circle', 5)
43  As = properties('square', 6)
44  Ar = properties('rectangle', 7, 8)
45  [Ac, Ic] = properties('circle', 5)
46  [As, Is] = properties('square', 6)
47  [Ar, Ir] = properties('rectangle', 7, 8)
48
49  function varargout = properties(varargin)
50  varargout{1} = 0;
51  switch varargin{1}
52      case 'circle'
53          if nargin == 2
54              r = varargin{2};
55              varargout{1} = pi*r^2;
56              if nargout == 2
57                  varargout{2} = pi*r^4/4;
58              end
59          end
60      case 'square'
61          if nargin == 2
62              a = varargin{2};
63              varargout{1} = a^2;
64              if nargout == 2
65                  varargout{2} = a^4/12;
66              end
67          end
68      case 'rectangle'
69          if nargin == 3
70              b = varargin{2};
71              h = varargin{3};
72              varargout{1} = b*h;
73              if nargout == 2
74                  varargout{2} = b*h^3/12;
75              end
76          end
77  end
78  end
```

```
Ac =
   78.5398
As =
    36
Ar =
    56
Ac =
   78.5398
Ic =
  490.8739
As =
    36
Is =
   108
Ar =
    56
Ir =
  298.6667
```

Table 4.2 Functions of Variable-Length Arguments	
Function	Description
`varargin`	Cell array containing the list of input arguments
`nargin`	Number of input arguments
`varargout`	Cell array containing the list of output arguments
`nargout`	Number of output arguments
Details and More: Help>MATLAB>Programming Scripts and Functions>Functions>Input and Output Arguments	

4.3 Structures

Example04_03.m

[2] Following commands demonstrate creations and manipulations of **structures**. A screen output is shown in [3-4].

```
clear
node.x = 1;
node.y = 1;
node.supportx = 0;
node.supporty = 0;
node.loadx = 0;
node.loady = -1000;
disp(node)
clear
node = struct('x',1,'y',1,'supportx',0,'supporty',0,'loadx',0,'loady',-1000)
Nodes(1) = node
Nodes(2) = struct('x',0,'y',0,'supportx',1,'supporty',1,'loadx',0,'loady',0);
Nodes(3) = struct('x',2,'y',0,'supportx',0,'supporty',1,'loadx',0,'loady',0);
disp(Nodes)
Nodes(1).x = 5.6;
disp(Nodes(1))
disp(node)
fieldnames(Nodes)
```

```
>> clear
>> node.x = 1;
>> node.y = 1;
>> node.supportx = 0;
>> node.supporty = 0;
>> node.loadx = 0;
>> node.loady = -1000;
>> disp(node)
           x: 1
           y: 1
    supportx: 0
    supporty: 0
       loadx: 0
```

[4] Screen output of Example04_03.m (Continued). →

```
>> clear
>> node = struct('x',1,'y',1,'supportx',0,'supporty',0,'loadx',0,'loady',-1000)
node =
  struct with fields:

           x: 1
           y: 1
    supportx: 0
    supporty: 0
       loadx: 0
       loady: -1000
>> Nodes(1) = node
Nodes =
  struct with fields:

           x: 1
           y: 1
    supportx: 0
    supporty: 0
       loadx: 0
       loady: -1000
>> Nodes(2) = struct('x',0,'y',0,'supportx',1,'supporty',1,'loadx',0,'loady',0);
>> Nodes(3) = struct('x',2,'y',0,'supportx',0,'supporty',1,'loadx',0,'loady',0);
>> disp(Nodes)
  1×3 struct array with fields:
    x
    y
    supportx
    supporty
    loadx
    loady
>> Nodes(1).x = 5.6;
>> disp(Nodes(1))
           x: 5.6000
           y: 1
    supportx: 0
    supporty: 0
       loadx: 0
       loady: -1000
>> disp(node)
           x: 1
           y: 1
    supportx: 0
    supporty: 0
       loadx: 0
       loady: -1000
>> fieldnames(Nodes)
ans =
  6×1 cell array
    'x'
    'y'
    'supportx'
    'supporty'
    'loadx'
    'loady'
```

Table 4.3 Structures	
Function	Description
`s = struct(field,v)`	Create structure, equivalent to `s.field = v`
`s = rmfield(s,field)`	Remove fields from structure
`names = fieldnames(s)`	Return a cell array of strings containing the names of the fields in a structure
`s = cell2struct(c)`	Convert cell array to structure
`s = table2struct(t)`	Convert table to structure
`c = struct2cell(s)`	Convert structure to cell
`t = struct2table(s)`	Convert structure to table
Details and More: Help>MATLAB>Language Fundamentals>Data Types>Structures	

4.4 Example: Statically Determinate Trusses (Version 3.0)

Example04_04.m: Truss 3.0

[1] This is a modification of the program Example03_15.m by replacing the **matrices** `Nodes` and `Members` with two **structure arrays**. The modification should enhance the readability of the program. Now, open Example03_15.m and save as Example04_04.m and modify the program as described in [2-11].

```
clear
Nodes = struct; Members = struct;
    disp(' 1. Input nodal coordinates')
    disp(' 2. Input connecting nodes of members')
    disp(' 3. Input three supports')
    disp(' 4. Input loads')
    disp(' 5. Print truss')
    disp(' 6. Solve truss')
    disp(' 7. Print results')
    disp(' 8. Save data')
    disp(' 9. Load data')
    disp('10. Quit')
while 1
    task = input('Enter the task number: ');
    switch task
        case 1
            Nodes = inputNodes(Nodes);
        case 2
            Members = inputMembers(Members);
        case 3
            Nodes = inputSupports(Nodes);
        case 4
            Nodes = inputLoads(Nodes);
        case 5
            printTruss(Nodes, Members)
        case 6
            [Nodes, Members] = solveTruss(Nodes, Members);
        case 7
            printResults(Nodes, Members)
        case 8
            saveAll(Nodes, Members)
        case 9
            [Nodes, Members] = loadAll;
        case 10
            break
    end
end
```

```
function output = inputNodes(Nodes)
while 1
    data = input('Enter [node, x, y] or 0 to stop: ');
    if data(1) == 0
        break
    else
        Nodes(data(1)).x = data(2);
        Nodes(data(1)).y = data(3);
    end
end
output = Nodes;
end

function output = inputMembers(Members)
m = 0;
while 1
    data = input('Enter [node1, node2] or 0 to stop: ');
    if data(1) == 0
        break
    else
        m = m+1;
        Members(m).node1 = data(1);
        Members(m).node2 = data(2);
    end
end
output = Members;
end

function output = inputSupports(Nodes)
for i = 1:size(Nodes,2)
    Nodes(i).supportx = 0;
    Nodes(i).supporty = 0;
end
for k = 1:3
    data = input('Enter [node, dir] (dir: ''x'' or ''y''): ');
    if data(2) == 'x'
        Nodes(data(1)).supportx = 1;
    elseif data(2) == 'y'
        Nodes(data(1)).supporty = 1;
    end
end
output = Nodes;
end

```

```
function output = inputLoads(Nodes)
for i = 1:size(Nodes,2)
    Nodes(i).loadx = 0;
    Nodes(i).loady = 0;
end
while 1
    data = input('Enter [node, load-x, load-y] or 0 to stop: ');
    if data(1) == 0
        break
    else
        Nodes(data(1)).loadx = data(2);
        Nodes(data(1)).loady = data(3);
    end
end
output = Nodes;
end

function printTruss(Nodes, Members)
if (size(fieldnames(Nodes),1)<6 || size(fieldnames(Members),1)<2)
    disp('Truss data not complete'); return
end
fprintf('\nNodal Data\n')
fprintf('Node        x        y  Support-x  Support-y   Load-x   Load-y\n')
for k = 1:size(Nodes,2)
    fprintf('%4.0f%9.2f%9.2f%11.0f%11.0f%9.0f%9.0f\n', ...
        k, Nodes(k).x, Nodes(k).y, ...
        Nodes(k).supportx, Nodes(k).supporty, ...
        Nodes(k).loadx, Nodes(k).loady)
end
fprintf('\nMember Data\n')
fprintf('Member  Node1    Node2\n')
for k = 1:size(Members,2)
    fprintf('%4.0f%9.0f%9.0f\n', k, Members(k).node1, Members(k).node2)
end
end
```

```
function printResults(Nodes, Members)
if (size(fieldnames(Nodes),1)<8 || size(fieldnames(Members),1)<3)
    disp('Results not available!'), return
end
fprintf('\nReaction Forces\n')
fprintf('Node  Reaction-x  Reaction-y\n')
for k = 1:size(Nodes,2)
    fprintf('%4.0f%12.2f%12.2f\n', k, Nodes(k).reactionx, Nodes(k).reactiony)
end
fprintf('\nMember Forces\n')
fprintf('Member     Force\n')
for k = 1:size(Members,2)
    fprintf('%4.0f%12.2f\n', k, Members(k).force)
end
end

function saveAll(Nodes, Members)
fileName = input('Enter file name (default Datafile): ', 's');
if isempty(fileName)
    fileName = 'Datafile';
end
save(fileName, 'Nodes', 'Members')
end

function [Nodes, Members] = loadAll
fileName = input('Enter file name (default Datafile): ', 's');
if isempty(fileName)
    fileName = 'Datafile';
end
load(fileName)
end

function [outNodes, outMembers] = solveTruss(Nodes, Members)
n = size(Nodes,2); m = size(Members,2);
if (m+3) < 2*n
    disp('Unstable!')
    outNodes = 0; outMembers = 0; return
elseif (m+3) > 2*n
    disp('Statically indeterminate!')
    outNodes = 0; outMembers = 0; return
end
```

```
A = zeros(2*n, 2*n); loads = zeros(2*n,1); nsupport = 0;
for i = 1:n
    for j = 1:m
        if Members(j).node1 == i || Members(j).node2 == i
            if Members(j).node1 == i
                n1 = i; n2 = Members(j).node2;
            elseif Members(j).node2 == i
                n1 = i; n2 = Members(j).node1;
            end
            x1 = Nodes(n1).x; y1 = Nodes(n1).y;
            x2 = Nodes(n2).x; y2 = Nodes(n2).y;
            L = sqrt((x2-x1)^2+(y2-y1)^2);
            A(2*i-1,j) = (x2-x1)/L;
            A(2*i,  j) = (y2-y1)/L;
        end
    end
    if (Nodes(i).supportx == 1)
        nsupport = nsupport+1;
        A(2*i-1,m+nsupport) = 1;
    end
    if (Nodes(i).supporty == 1)
        nsupport = nsupport+1;
        A(2*i, m+nsupport) = 1;
    end
    loads(2*i-1) = -Nodes(i).loadx;
    loads(2*i)   = -Nodes(i).loady;
end
forces = A\loads;
for j = 1:m
    Members(j).force = forces(j);
end
nsupport = 0;
for i = 1:n
    Nodes(i).reactionx = 0;
    Nodes(i).reactiony = 0;
    if (Nodes(i).supportx == 1)
        nsupport = nsupport+1;
        Nodes(i).reactionx = forces(m+nsupport);
    end
    if (Nodes(i).supporty == 1)
        nsupport = nsupport+1;
        Nodes(i).reactiony = forces(m+nsupport);
    end
end
outNodes = Nodes; outMembers = Members;
disp('Solved successfully.')
end
```

4.5 Tables

Example04_05.m

[2] The following commands demonstrate the creation and manipulation of **tables**. A screen output is shown in [3-4]. →

```
clear
x = [1, 0, 2]'; y = [1, 0, 0]';
supportx = [0, 1, 0]'; supporty = [0, 1, 1]';
loadx = [0, 0 0]'; loady = [-1000, 0, 0]';
Nodes = table(x, y, supportx, supporty, loadx, loady)
Nodes.Properties
Nodes.Properties.RowNames = {'top', 'left', 'right'};
disp(Nodes)
size(Nodes)
Nodes = sortrows(Nodes, {'x', 'y'})
node = Nodes(2,:)
Nodes(4,:) = array2table([2, 2, 0, 0, 100, 200]);
Nodes.Properties.RowNames{4} = 'node4';
Nodes(5,:) = cell2table({0, 2, 0, 0, 0, 0});
n = struct('x', 1.5, 'y', 0.5, ...
    'supportx', 0, 'supporty', 0, 'loadx', 0, 'loady',0);
Nodes = [Nodes; struct2table(n)];
Nodes(6,5) = array2table(300);
class(Nodes(6,5))
Nodes.loady(6) = 150;
class(Nodes.loadx(6))
disp(Nodes)
Nodes(4:6,:) = [];
disp(Nodes)
Nodes(1:3,:) = [];
size(Nodes)
```

```
>> clear
>> x = [1, 0, 2]'; y = [1, 0, 0]';
>> supportx = [0, 1, 0]'; supporty = [0, 1, 1]';
>> loadx = [0, 0 0]'; loady = [-1000, 0, 0]';
>> Nodes = table(x, y, supportx, supporty, loadx, loady)
Nodes =
  3×6 table
    x    y    supportx    supporty    loadx    loady
    _    _    ________    ________    _____    _____
    1    1    0           0           0        -1000
    0    0    1           1           0            0
    2    0    0           1           0            0
>> Nodes.Properties
ans =
  struct with fields:

             Description: ''
                UserData: []
          DimensionNames: {'Row'  'Variables'}
           VariableNames: {'x'  'y'  'supportx'  'supporty'  'loadx'  'loady'}
    VariableDescriptions: {}
           VariableUnits: {}
                RowNames: {}
>> Nodes.Properties.RowNames = {'top', 'left', 'right'};
>> disp(Nodes)
             x    y    supportx    supporty    loadx    loady
             _    _    ________    ________    _____    _____
    top      1    1    0           0           0        -1000
    left     0    0    1           1           0            0
    right    2    0    0           1           0            0
>> size(Nodes)
ans =
     3     6
>> Nodes = sortrows(Nodes, {'x', 'y'})
Nodes =
  3×6 table
             x    y    supportx    supporty    loadx    loady
             _    _    ________    ________    _____    _____
    left     0    0    1           1           0            0
    top      1    1    0           0           0        -1000
    right    2    0    0           1           0            0
>> node = Nodes(2,:)
node =
  1×6 table
           x    y    supportx    supporty    loadx    loady
           _    _    ________    ________    _____    _____
    top    1    1    0           0           0        -1000
```

```
>> Nodes(4,:) = array2table([2, 2, 0, 0, 100, 200]);
>> Nodes.Properties.RowNames{4} = 'node4';
>> Nodes(5,:) = cell2table({0, 2, 0, 0, 0, 0});
>> n = struct('x', 1.5, 'y', 0.5, ...
    'supportx', 0, 'supporty', 0, 'loadx', 0, 'loady',0);
>> Nodes = [Nodes; struct2table(n)];
>> Nodes(6,5) = array2table(300);
>> class(Nodes(6,5))
ans =
    'table'
>> Nodes.loady(6) = 150;
>> class(Nodes.loadx(6))
ans =
    'double'
>> disp(Nodes)
                 x      y     supportx    supporty    loadx    loady
                ___    ___    ________    ________    _____    _____

    left         0      0     1           1             0          0
    top          1      1     0           0             0      -1000
    right        2      0     0           1             0          0
    node4        2      2     0           0           100        200
    Row5         0      2     0           0             0          0
    Row6       1.5    0.5     0           0           300        150
>> Nodes(4:6,:) = [];
>> disp(Nodes)
                 x    y    supportx    supporty    loadx    loady
                 _    _    ________    ________    _____    _____

    left         0    0    1           1           0            0
    top          1    1    0           0           0        -1000
    right        2    0    0           1           0            0
>> Nodes(1:3,:) = [];
>> size(Nodes)
ans =
     0     6
```

Table 4.5 Tables	
Function	Description
`t = table(v1,v2,...)`	Create table from variables
`t = array2table(a)`	Convert array to table
`t = cell2table(c)`	Convert cell array to table
`t = struct2table(s)`	Convert structure to table
`a = table2array(t)`	Convert table to array
`c = table2cell(t)`	Convert table to cell
`s = table2struct(t)`	Convert table to structure
`t = sortrows(t)`	Sort rows of array or table
`T.Properties`	Properties of table T
Details and More: Help>MATLAB>Language Fundamentals>Data Types>Tables	

4.6 Conversion of Cell Arrays

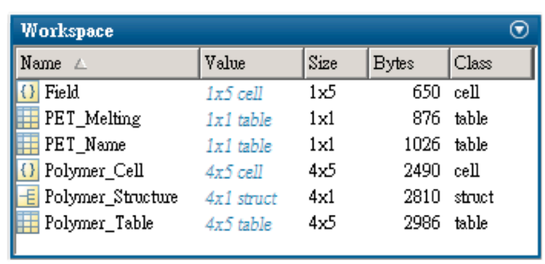
Name	Abbreviation	Melting Temperature (°C)	Crystallization Temperature (°C)	Density (g/cm^3)
Polyethylene	PE	135	56	0.96
Polypropylene	PP	171	86	0.95
Polyoxymethylene	POM	180	90	1.42
Polyethylene terephthalate	PET	266	158	1.38

Example04_06.m: Polymer Database

[2] This script first creates a **cell array** to store the information (lines 2-5), then converts the **cell array** to a **structure array** using the function `cell2struct` (lines 9-10), and finally converts the **cell array** to a **table** using the function `cell2table` (line 14). In each stage, it displays PET's name and melting temperature (lines 6-7, 11-12, and 15-18) to demonstrate the access of data. →

```
 1  clear
 2  Polymer_Cell = {'Polyethylene',               'PE',  135,  56, 0.96;
 3                  'Polypropylene',              'PP',  171,  86, 0.95;
 4                  'Polyoxymethylene',           'POM', 180,  90, 1.42;
 5                  'Polyethylene terephthalate', 'PET', 266, 158, 1.38};
 6  PET_Name = Polymer_Cell{4,1}
 7  PET_Melting = Polymer_Cell{4,3}
 8
 9  Field = {'Name', 'Abbreviation', 'Melting', 'Crystallization', 'Density'};
10  Polymer_Structure = cell2struct(Polymer_Cell, Field, 2);
11  PET_Name = Polymer_Structure(4).Name
12  PET_Melting = Polymer_Structure(4).Melting
13
14  Polymer_Table = cell2table(Polymer_Cell, 'VariableNames', Field);
15  PET_Name = Polymer_Table.Name(4)
16  PET_Melting = Polymer_Table.Melting(4)
17  PET_Name = Polymer_Table(4,1)
18  PET_Melting = Polymer_Table(4,3)
```

```
>> clear
>> Polymer_Cell = {'Polyethylene',                'PE',  135,  56, 0.96;
                   'Polypropylene',               'PP',  171,  86, 0.95;
                   'Polyoxymethylene',            'POM', 180,  90, 1.42;
                   'Polyethylene terephthalate', 'PET', 266, 158, 1.38};
>> PET_Name = Polymer_Cell{4,1}
PET_Name =
    'Polyethylene terephthalate'
>> PET_Melting = Polymer_Cell{4,3}
PET_Melting =
   266
>>
>> Field = {'Name', 'Abbreviation', 'Melting', 'Crystallization', 'Density'};
>> Polymer_Structure = cell2struct(Polymer_Cell, Field, 2);
>> PET_Name = Polymer_Structure(4).Name
PET_Name =
    'Polyethylene terephthalate'
>> PET_Melting = Polymer_Structure(4).Melting
PET_Melting =
   266
>>
>> Polymer_Table = cell2table(Polymer_Cell, 'VariableNames', Field);
>> PET_Name = Polymer_Table.Name(4)
PET_Name =
  cell
    'Polyethylene terephthalate'
>> PET_Melting = Polymer_Table.Melting(4)
PET_Melting =
   266
>> PET_Name = Polymer_Table(4,1)
PET_Name =
  table
                 Name
    ____________________________
    'Polyethylene terephthalate'
>> PET_Melting = Polymer_Table(4,3)
PET_Melting =
  table
    Melting
    _______
    266
```



Workspace

Name	Value	Size	Bytes	Class
Field	1x5 cell	1x5	650	cell
PET_Melting	1x1 table	1x1	876	table
PET_Name	1x1 table	1x1	1026	table
Polymer_Cell	4x5 cell	4x5	2490	cell
Polymer_Structure	4x1 struct	4x1	2810	struct
Polymer_Table	4x5 table	4x5	2986	table

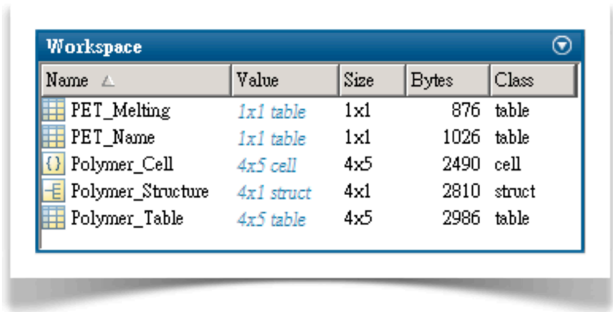
4.7 Conversion of Structure Arrays

Example04_07.m: Polymer Database

[2] This script creates a **structure array** storing the polymer data (lines 2-21), converts the **structure array** to a **cell array** using the function `struct2cell` (line 25), and converts the **structure array** to a **table** using the function `struct2table` (line 30). In each stage, it displays PET's name and melting temperature (lines 22-23, 27-28, and 31-34) to demonstrate the access of data. →

```
 1  clear
 2  Polymer_Structure = [struct('Name', 'Polyethylene',              ...
 3                              'Abbreviation', 'PE',                 ...
 4                              'Melting', 135,                       ...
 5                              'Crystallization', 56,                ...
 6                              'Density', 0.96);
 7                       struct('Name', 'Polypropylene',              ...
 8                              'Abbreviation', 'PP',                 ...
 9                              'Melting', 171,                       ...
10                              'Crystallization', 86,                ...
11                              'Density', 0.95);
12                       struct('Name', 'Polyoxymethylene',           ...
13                              'Abbreviation', 'POM',                ...
14                              'Melting', 180,                       ...
15                              'Crystallization', 90,                ...
16                              'Density', 1.42);
17                       struct('Name', 'Polyethylene terephthalate', ...
18                              'Abbreviation', 'PET',                ...
19                              'Melting', 266,                       ...
20                              'Crystallization', 158',              ...
21                              'Density', 1.38)];
22  PET_Name = Polymer_Structure(4).Name
23  PET_Melting = Polymer_Structure(4).Melting
24
25  Polymer_Cell = struct2cell(Polymer_Structure);
26  Polymer_Cell = Polymer_Cell';
27  PET_Name = Polymer_Cell{4,1}
28  PET_Melting = Polymer_Cell{4,3}
29
30  Polymer_Table = struct2table(Polymer_Structure);
31  PET_Name = Polymer_Table.Name(4)
32  PET_Melting = Polymer_Table.Melting(4)
33  PET_Name = Polymer_Table(4,1)
34  PET_Melting = Polymer_Table(4,3)
```

```
>> clear
>> Polymer_Structure = [struct('Name', 'Polyethylene',                ...
                               'Abbreviation', 'PE',                  ...
                               'Melting', 135,                        ...
                               'Crystallization', 56,                 ...
                               'Density', 0.96);
                        struct('Name', 'Polypropylene',               ...
                               'Abbreviation', 'PP',                  ...
                               'Melting', 171,                        ...
                               'Crystallization', 86,                 ...
                               'Density', 0.95);
                        struct('Name', 'Polyoxymethylene',            ...
                               'Abbreviation', 'POM',                 ...
                               'Melting', 180,                        ...
                               'Crystallization', 90,                 ...
                               'Density', 1.42);
                        struct('Name', 'Polyethylene terephthalate',  ...
                               'Abbreviation', 'PET',                 ...
                               'Melting', 266,                        ...
                               'Crystallization', 158',               ...
                               'Density', 1.38)];
>> PET_Name = Polymer_Structure(4).Name
PET_Name =
    'Polyethylene terephthalate'
>> PET_Melting = Polymer_Structure(4).Melting
PET_Melting =
   266
>>
>> Polymer_Cell = struct2cell(Polymer_Structure);
>> Polymer_Cell = Polymer_Cell';
>> PET_Name = Polymer_Cell{4,1}
PET_Name =
    'Polyethylene terephthalate'
>> PET_Melting = Polymer_Cell{4,3}
PET_Melting =
   266
>>
>> Polymer_Table = struct2table(Polymer_Structure);
>> PET_Name = Polymer_Table.Name(4)
PET_Name =
  cell
    'Polyethylene terephthalate'
>> PET_Melting = Polymer_Table.Melting(4)
PET_Melting =
   266
>> PET_Name = Polymer_Table(4,1)
PET_Name =
  table
                Name
    ____________________________
    'Polyethylene terephthalate'
>> PET_Melting = Polymer_Table(4,3)
PET_Melting =
  table
    Melting
    _______
    266
```


Workspace

Name	Value	Size	Bytes	Class
PET_Melting	1x1 table	1x1	876	table
PET_Name	1x1 table	1x1	1026	table
Polymer_Cell	4x5 cell	4x5	2490	cell
Polymer_Structure	4x1 struct	4x1	2810	struct
Polymer_Table	4x5 table	4x5	2986	table

4.8 Conversion of Tables

Example04_08.m: Polymer Database

[2] This script creates a **table** storing the polymer data (lines 2-8), converts the **table** to a **cell array** using the function `table2cell` (line 14), and converts the **table** to a **structure array** using the function `table2struct` (line 18). In each stage, it displays PET's name and melting temperature (lines 9-12, 15-16, and 19-20) to demonstrate the access of data.

```
 1   clear
 2   Name = {'Polyethylene', 'Polypropylene', 'Polyoxymethylene', ...
 3           'Polyethylene terephthalate'}';
 4   Abbreviation = {'PE', 'PP', 'POM', 'PET'}';
 5   Melting = [135, 171, 180, 266]';
 6   Crystallization = [56, 86, 90, 1585]';
 7   Density = [0.96, 0.95, 1.42, 1.38]';
 8   Polymer_Table = table(Name,Abbreviation,Melting,Crystallization,Density);
 9   PET_Name = Polymer_Table.Name(4)
10   PET_Melting = Polymer_Table.Melting(4)
11   PET_Name = Polymer_Table(4,1)
12   PET_Melting = Polymer_Table(4,3)
13
14   Polymer_Cell = table2cell(Polymer_Table);
15   PET_Name = Polymer_Cell{4,1}
16   PET_Melting = Polymer_Cell{4,3}
17
18   Polymer_Structure = table2struct(Polymer_Table);
19   PET_Name = Polymer_Structure(4).Name
20   PET_Melting = Polymer_Structure(4).Melting
```

Workspace

Name	Value	Size	Bytes	Class
Abbreviation	4x1 cell	4x1	468	cell
Crystallization	[56;86;90;1585]	4x1	32	double
Density	[0.9600;0.9500;1.4200;1.3800]	4x1	32	double
Melting	[135;171;180;266]	4x1	32	double
Name	4x1 cell	4x1	582	cell
PET_Melting	266	1x1	8	double
PET_Name	'Polyethylene terephthalate'	1x26	52	char
Polymer_Cell	4x5 cell	4x5	2490	cell
Polymer_Structure	4x1 struct	4x1	2810	struct
Polymer_Table	4x5 table	4x5	2986	table

```
>> clear
>> Name = {'Polyethylene', 'Polypropylene', 'Polyoxymethylene', ...
           'Polyethylene terephthalate'}';
>> Abbreviation = {'PE', 'PP', 'POM', 'PET'}';
>> Melting = [135, 171, 180, 266]';
>> Crystallization = [56, 86, 90, 1585]';
>> Density = [0.96, 0.95, 1.42, 1.38]';
>> Polymer_Table = table(Name,Abbreviation,Melting,Crystallization,Density);
>> PET_Name = Polymer_Table.Name(4)
PET_Name =
  cell
    'Polyethylene terephthalate'
>> PET_Melting = Polymer_Table.Melting(4)
PET_Melting =
   266
>> PET_Name = Polymer_Table(4,1)
PET_Name =
  table
                Name
    ____________________________
    'Polyethylene terephthalate'
>> PET_Melting = Polymer_Table(4,3)
PET_Melting =
  table
    Melting
    _______
    266
>>
>> Polymer_Cell = table2cell(Polymer_Table);
>> PET_Name = Polymer_Cell{4,1}
PET_Name =
    'Polyethylene terephthalate'
>> PET_Melting = Polymer_Cell{4,3}
PET_Melting =
   266
>>
>> Polymer_Structure = table2struct(Polymer_Table);
>> PET_Name = Polymer_Structure(4).Name
PET_Name =
    'Polyethylene terephthalate'
>> PET_Melting = Polymer_Structure(4).Melting
PET_Melting =
   266
```

4.9 User-Defined Classes

Poly.m

[2] Type the following statements and save in the **Current Folder** as **Poly.m** (the file name must be consistent with the class name, specified in line 1). The class Poly is an implementation of the polynomial (*Wikipedia>Polynomial*), including its representing data structure and some operations on the polynomial. This is a simple demonstration of user-defined class. We'll show you the application of this class in [4]. (Continued at [3], next page.) →

```
 1  classdef Poly
 2      properties
 3          coef = zeros(1,99);
 4      end
 5      methods
 6          function p = Poly(varargin)
 7              for k = 1:nargin
 8                  p.coef(nargin-k+1) = varargin{k};
 9              end
10          end
11          function disp(p)
12              for k = 99:-1:3
13                  if p.coef(k)
14                      fprintf('%+fx^%d', p.coef(k), k-1);
15                  end
16              end
17              if p.coef(2)
18                  fprintf('%+fx', p.coef(2));
19              end
20              fprintf('%+f\n', p.coef(1))
21          end
22          function p = plus(p1, p2)
23              p = Poly;
24              p.coef = p1.coef+p2.coef;
25          end
26          function p = minus(p1, p2)
27              p = Poly;
28              p.coef = p1.coef-p2.coef;
29          end
30          function p = uminus(p1)
31              p = Poly;
32              p.coef = -p1.coef;
33          end
```

[3] Poly.m (Continued)

```
        function p = uplus(p1)
            p = Poly;
            p.coef = p1.coef;
        end
        function p = mtimes(p1, p2)
            p = Poly;
            for i = 1:99
                for j = 1:99
                    if p1.coef(i) && p2.coef(j)
                        p.coef(i+j-1) = p.coef(i+j-1)+p1.coef(i)*p2.coef(j);
                    end
                end
            end
        end
        function output = value(p, x)
            output = zeros(1,length(x));
            for k = 1:99
                if p.coef(k)
                    output = output + p.coef(k)*x.^(k-1);
                end
            end
        end
    end
end
```

```
>> clear
>> a = Poly(3,2,1)
a =
+3.000000x^2+2.000000x+1.000000
>> b = Poly(5,6)
b =
+5.000000x+6.000000
>> c = Poly(8)
c =
+8.000000
>> d = a+b
d =
+3.000000x^2+7.000000x+7.000000
>> e = -a-b*(-c+a)
e =
-15.000000x^3-31.000000x^2+21.000000x+41.000000
>> value(e, 2.5)
ans =
 -334.6250
>> x = linspace(-3,3);
>> y = value(e,x);
>> plot(x,y)
```

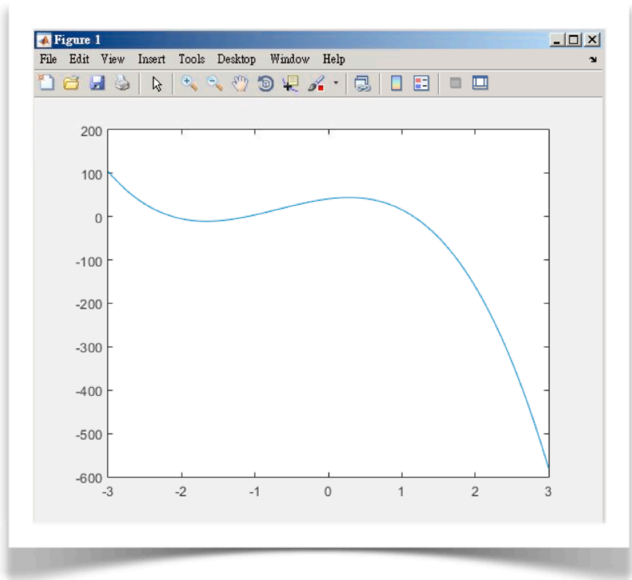


Figure 1
File Edit View Insert Tools Desktop Window Help
200
100
0
-100
-200
-300
-400
-500
-600
-3
-2
-1
0
1
2
3

Differentiation and Integration of a Polynomial

[9] You may implement the differentiation and integration of a polynomial by adding the following two functions into the **methods** section of the class `Poly`:

```
function p = diff(p1)
    p = Poly;
    for k = 2:99
        if p1.coef(k)
            p.coef(k-1) = p1.coef(k)*(k-1);
        end
    end
end
function p = int(p1)
    p = Poly;
    for k = 1:99
        if p1.coef(k)
            p.coef(k+1) = p1.coef(k)/k;
        end
    end
end
```

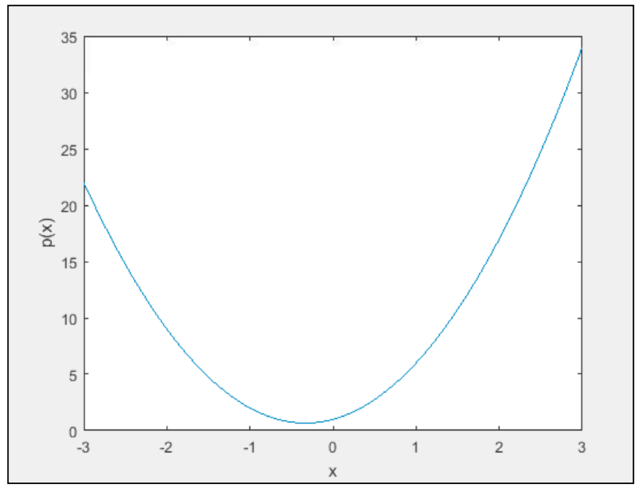
And you may test-run the new version of `Poly` by continuing the commands in Example04_09.m ([4], page 201):

```
>> f = int(e)
f =
-3.750000x^4-10.333333x^3+10.500000x^2+41.000000x+0.000000
>> g = diff(f)
g =
-15.000000x^3-31.000000x^2+21.000000x+41.000000
```

4.10 Additional Exercise Problems

Name	Symbol	Atomic Number	Atomic Mass
Carbon	C	6	12.011
Helium	He	2	4.003
Hydrogen	H	1	1.008
Nitrogen	N	7	14.007
Oxygen	O	8	15.999

Name	Symbol	Atomic Number	Atomic Mass
Sodium	Na	11	22.990
Chlorine	Cl	17	35.453

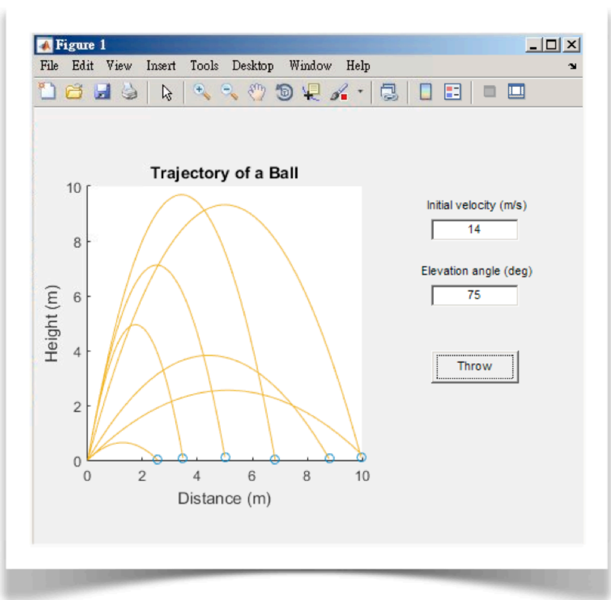

p(x)
x

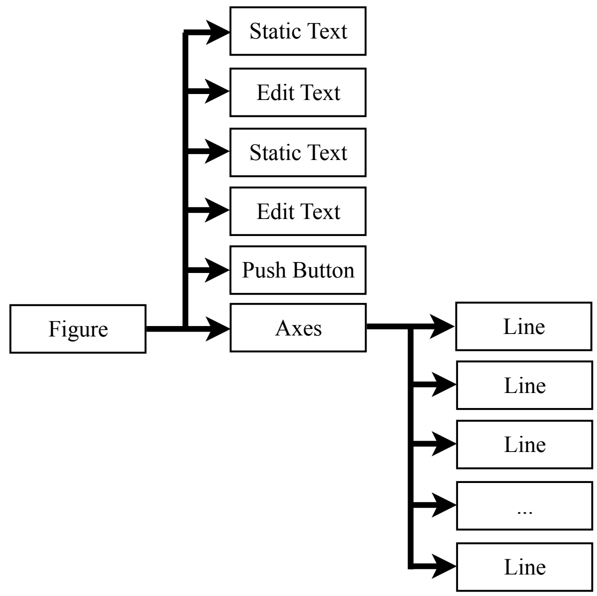
Chapter 5

Data Visualization: Plots

Visual perception is the most efficient way of understanding data. MATLAB provides many forms of plots to aid engineers in presenting their data in various visual forms. Familiarizing yourself with these plotting techniques will facilitate your presentation of data.

5.1 Graphics Objects and Parent-Children Relationship

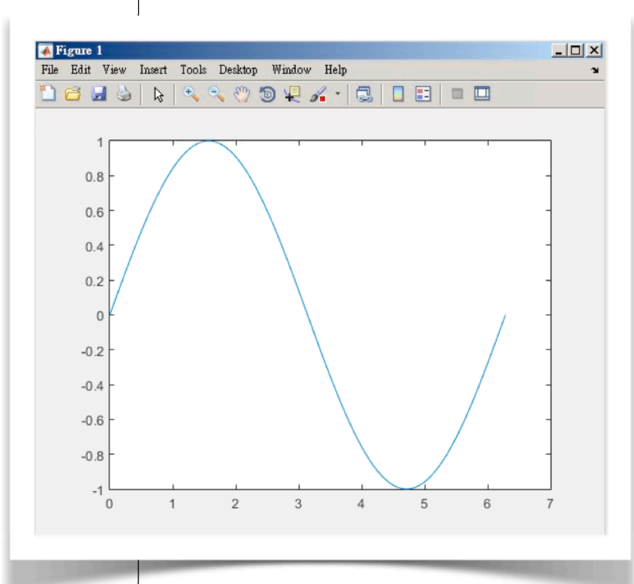




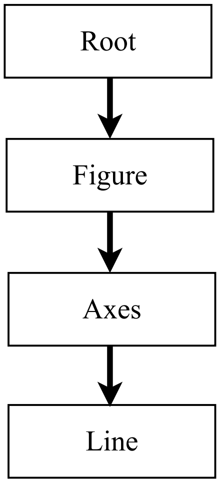
Example05_01a.m

[2] The following commands create some graphics objects and demonstrate the parent-children relationship among these graphic objects.

```
clear
x = linspace(0,2*pi);
y = sin(x);
hCurve = plot(x,y);
hCurve.Parent
hAxes = hCurve.Parent;
hAxes.Parent
hFigure = hAxes.Parent;
hFigure.Parent
hRoot = hFigure.Parent;
hRoot.Parent
hRoot.Children
hFigure.Children
hAxes.Children
hCurve.Children
delete(hCurve)
delete(hAxes)
delete(hFigure)
```



```
>> clear
>> x = linspace(0,2*pi);
>> y = sin(x);
>> hCurve = plot(x,y);
>> hCurve.Parent
ans =
  Axes with properties:

             XLim: [0 7]
             YLim: [-1 1]
           XScale: 'linear'
           YScale: 'linear'
    GridLineStyle: '-'
         Position: [0.1300 0.1100 0.7750 0.8150]
            Units: 'normalized'

  Show all properties
>> hAxes = hCurve.Parent;
>> hAxes.Parent
ans =
  Figure (1) with properties:

      Number: 1
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [1000 918 560 420]
       Units: 'pixels'

  Show all properties
>> hFigure = hAxes.Parent;
>> hFigure.Parent
ans =
  Graphics Root with properties:

          CurrentFigure: [1×1 Figure]
    ScreenPixelsPerInch: 72
             ScreenSize: [1 1 2560 1440]
       MonitorPositions: [1 1 2560 1440]
                  Units: 'pixels'

  Show all properties
>> hRoot = hFigure.Parent;
>> hRoot.Parent
ans =
  0×0 empty GraphicsPlaceholder array.
```



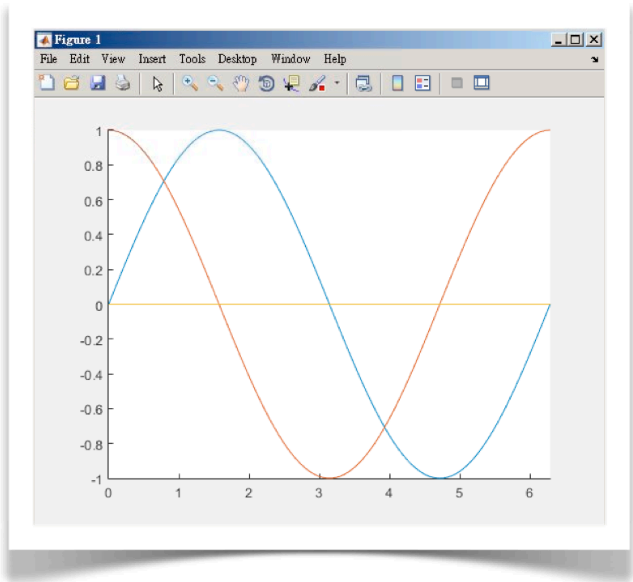

```
>> hRoot.Children
ans =
  Figure (1) with properties:

      Number: 1
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [1000 918 560 420]
       Units: 'pixels'

  Show all properties
>> hFigure.Children
ans =
  Axes with properties:

             XLim: [0 7]
             YLim: [-1 1]
           XScale: 'linear'
           YScale: 'linear'
    GridLineStyle: '-'
         Position: [0.1300 0.1100 0.7750 0.8150]
            Units: 'normalized'

  Show all properties
>> hAxes.Children
ans =
  Line with properties:

              Color: [0 0.4470 0.7410]
          LineStyle: '-'
          LineWidth: 0.5000
             Marker: 'none'
         MarkerSize: 6
    MarkerFaceColor: 'none'
              XData: [1×100 double]
              YData: [1×100 double]
              ZData: [1×0 double]

  Show all properties
>> hCurve.Children
ans =
  0×0 empty GraphicsPlaceholder array.
>> delete(hCurve)
>> delete(hAxes)
>> delete(hFigure)
```

Example05_01b.m

[7] In Example05_01a.m, when a **Line** is created, a **Figure** and an **Axes** are automatically created. It is possible to create **Figures** and **Axes** manually. The following commands demonstrate manual creation of a **Figure** and an **Axes** before creation of three **Line** objects as children of the **Axes**.

```
clear
x = linspace(0,2*pi);
figure
axes('XLim', [0,2*pi], 'YLim', [-1,1])
hold on
plot(x, sin(x), x, cos(x))
plot([0,2*pi],[0,0])
hAxes = gca;
hCurve = hAxes.Children
delete(hCurve(1))
delete(hCurve(2))
delete(hAxes)
delete(gcf)
```

```
>> clear
>> x = linspace(0,2*pi);
>> figure
>> axes('XLim', [0,2*pi], 'YLim', [-1,1])
>> hold on
>> plot(x, sin(x), x, cos(x))
>> plot([0,2*pi],[0,0])
>> hAxes = gca;
>> hCurve = hAxes.Children
hCurve =
  3×1 Line array:

  Line
  Line
  Line
>> delete(hCurve(1))
>> delete(hCurve(2))
>> delete(hAxes)
>> delete(gcf)
```



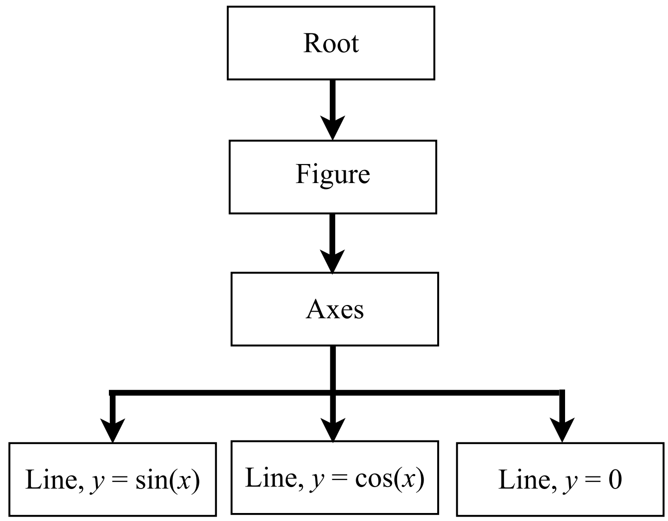
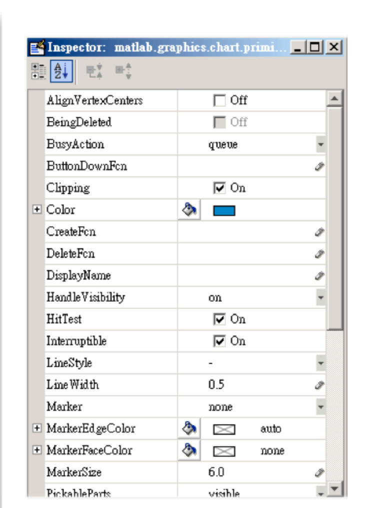


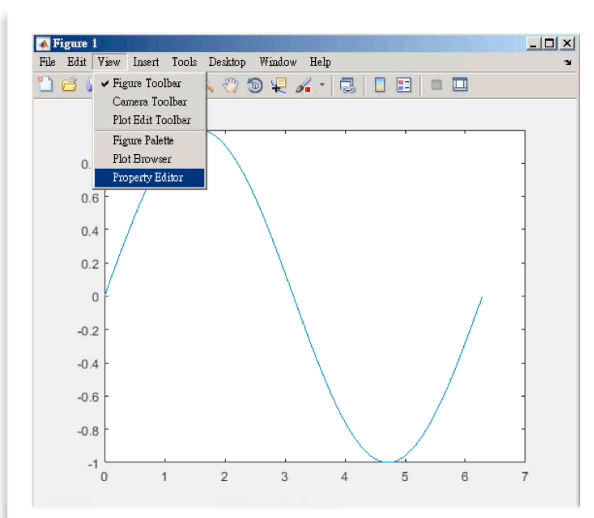
Table 5.1 Graphics Objects

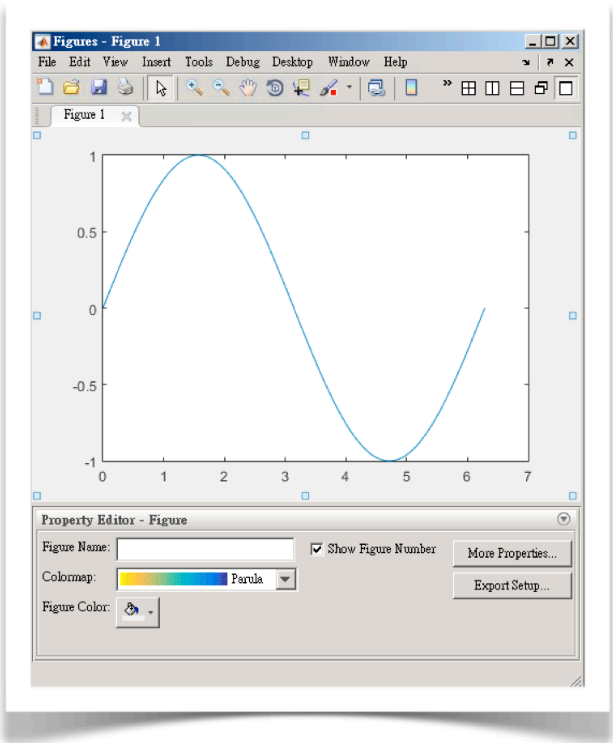
Function	Description
`hf = figure`	Create figure window
`ha = axes`	Create axes object
`hr = groot`	Get handle of root object
`hf = gcf`	Get handle of current figure
`ha = gca`	Get handle of current axes
`ho = gco`	Get handle of current object
`v = get(h,property)`	Get graphics object properties
`set(h,name,value)`	Set graphics object properties
`delete(h)`	Delete objects

Details and More: Help>MATLAB>Graphics>Graphics Objects

5.2 Graphics Objects Properties







[8] Output of `get(hCurve)`.

```
            Alphamap: [1x64 double]
    AlignVertexCenters: 'off'
          Annotation: [1x1 ...]
        BeingDeleted: 'off'
          BusyAction: 'queue'
       ButtonDownFcn: ''
            Children: []
            Clipping: 'on'
               Color: [0 0.4470 0.7410]
           CreateFcn: ''
           DeleteFcn: ''
         DisplayName: ''
    HandleVisibility: 'on'
             HitTest: 'on'
       Interruptible: 'on'
           LineStyle: '-'
           LineWidth: 0.5000
              Marker: 'none'
     MarkerEdgeColor: 'auto'
     MarkerFaceColor: 'none'
          MarkerSize: 6
              Parent: [1x1 Axes]
       PickableParts: 'visible'
            Selected: 'off'
  SelectionHighlight: 'on'
                 Tag: ''
                Type: 'line'
       UIContextMenu: []
            UserData: []
             Visible: 'on'
               XData: [1x100 double]
           XDataMode: 'manual'
         XDataSource: ''
               YData: [1x100 double]
         YDataSource: ''
               ZData: [1x0 double]
         ZDataSource: ''
```

[9] Output of `get(groot)`. →

```
        CallbackObject: [0×0 GraphicsPlaceholder]
              Children: [0×0 GraphicsPlaceholder]
         CurrentFigure: [0×0 GraphicsPlaceholder]
     FixedWidthFontName: 'Courier New'
      HandleVisibility: 'on'
      MonitorPositions: [1 1 2560 1440]
                Parent: [0×0 GraphicsPlaceholder]
       PointerLocation: [2232 543]
           ScreenDepth: 32
  ScreenPixelsPerInch: 72
            ScreenSize: [1 1 2560 1440]
      ShowHiddenHandles: 'off'
                   Tag: ''
                  Type: 'root'
                 Units: 'pixels'
              UserData: []
```

[10] Output of `get(gcf)`.

```
             Alphamap: [1x64 double]
         BeingDeleted: 'off'
           BusyAction: 'queue'
        ButtonDownFcn: ''
             Children: [1x1 Axes]
             Clipping: 'on'
      CloseRequestFcn: 'closereq'
                Color: [0.94 0.94 0.94]
             Colormap: [64x3 double]
            CreateFcn: ''
          CurrentAxes: [1x1 Axes]
     CurrentCharacter: ''
        CurrentObject: []
         CurrentPoint: [0 0]
            DeleteFcn: ''
         DockControls: 'on'
             FileName: ''
    GraphicsSmoothing: 'on'
     HandleVisibility: 'on'
        IntegerHandle: 'on'
        Interruptible: 'on'
       InvertHardcopy: 'on'
          KeyPressFcn: ''
        KeyReleaseFcn: ''
              MenuBar: 'figure'
                 Name: ''
             NextPlot: 'add'
               Number: 1
          NumberTitle: 'on'
     PaperOrientation: 'portrait'
        PaperPosition: [0.2500 2.5000 8 6]
    PaperPositionMode: 'manual'
            PaperSize: [8.5000 11]
            PaperType: 'usletter'
           PaperUnits: 'inches'
               Parent: [1x1 Root]
              Pointer: 'arrow'
    PointerShapeCData: [16x16 double]
  PointerShapeHotSpot: [1 1]
             Position: [520 378 560 420]
             Renderer: 'opengl'
         RendererMode: 'auto'
               Resize: 'on'
        SelectionType: 'normal'
       SizeChangedFcn: ''
                  Tag: ''
              ToolBar: 'auto'
                 Type: 'figure'
        UIContextMenu: []
                Units: 'pixels'
             UserData: []
              Visible: 'on'
  WindowButtonDownFcn: ''
WindowButtonMotionFcn: ''
    WindowButtonUpFcn: ''
    WindowKeyPressFcn: ''
  WindowKeyReleaseFcn: ''
 WindowScrollWheelFcn: ''
          WindowStyle: 'normal'
```

[11] Output of `get(gca)` (Continued on next page). →

```
                   ALim: [0 1]
               ALimMode: 'auto'
 ActivePositionProperty: 'outerposition'
      AmbientLightColor: [1 1 1]
           BeingDeleted: 'off'
                    Box: 'on'
               BoxStyle: 'back'
             BusyAction: 'queue'
          ButtonDownFcn: ''
                   CLim: [0 1]
               CLimMode: 'auto'
         CameraPosition: [3.5000 0 17.3205]
     CameraPositionMode: 'auto'
           CameraTarget: [3.5000 0 0]
       CameraTargetMode: 'auto'
         CameraUpVector: [0 1 0]
     CameraUpVectorMode: 'auto'
        CameraViewAngle: 6.6086
    CameraViewAngleMode: 'auto'
               Children: [1x1 Line]
               Clipping: 'on'
          ClippingStyle: '3dbox'
                  Color: [1 1 1]
             ColorOrder: [7x3 double]
        ColorOrderIndex: 2
              CreateFcn: ''
           CurrentPoint: [2x3 double]
        DataAspectRatio: [3.5000 1 1]
    DataAspectRatioMode: 'auto'
              DeleteFcn: ''
              FontAngle: 'normal'
               FontName: 'Helvetica'
               FontSize: 10
          FontSmoothing: 'on'
              FontUnits: 'points'
             FontWeight: 'normal'
              GridAlpha: 0.1500
          GridAlphaMode: 'auto'
              GridColor: [0.15 0.15 0.15]
          GridColorMode: 'auto'
          GridLineStyle: '-'
       HandleVisibility: 'on'
                HitTest: 'on'
          Interruptible: 'on'
LabelFontSizeMultiplier: 1.1000
                  Layer: 'bottom'
         LineStyleOrder: '-'
    LineStyleOrderIndex: 1
              LineWidth: 0.5000
         MinorGridAlpha: 0.2500
     MinorGridAlphaMode: 'auto'
         MinorGridColor: [0.10 0.10 0.10]
     MinorGridColorMode: 'auto'
     MinorGridLineStyle: ':'
               NextPlot: 'replace'
          OuterPosition: [0 0 1 1]
                 Parent: [1x1 Figure]
```

[12] Output of `get(gca)` (Continued).

```
           PickableParts: 'visible'
      PlotBoxAspectRatio: [1 0.7903 0.7903]
  PlotBoxAspectRatioMode: 'auto'
                Position: [0.1300 0.1100 0.7750 0.8150]
              Projection: 'orthographic'
                Selected: 'off'
      SelectionHighlight: 'on'
              SortMethod: 'childorder'
                     Tag: ''
                 TickDir: 'in'
             TickDirMode: 'auto'
    TickLabelInterpreter: 'tex'
              TickLength: [0.0100 0.0250]
              TightInset: [0.0506 0.0532 0.0071 0.0202]
                   Title: [1x1 Text]
 TitleFontSizeMultiplier: 1.1000
         TitleFontWeight: 'bold'
                    Type: 'axes'
           UIContextMenu: []
                   Units: 'normalized'
                UserData: []
                    View: [0 90]
                 Visible: 'on'
           XAxisLocation: 'bottom'
                  XColor: [0.1500 0.1500 0.1500]
              XColorMode: 'auto'
                    XDir: 'normal'
                   XGrid: 'off'
                  XLabel: [1x1 Text]
                    XLim: [0 7]
                XLimMode: 'auto'
              XMinorGrid: 'off'
              XMinorTick: 'off'
                  XScale: 'linear'
                   XTick: [0 1 2 3 4 5 6 7]
              XTickLabel: {8x1 cell}
          XTickLabelMode: 'auto'
      XTickLabelRotation: 0
               XTickMode: 'auto'
           YAxisLocation: 'left'
                  YColor: [0.1500 0.1500 0.1500]
              YColorMode: 'auto'
                    YDir: 'normal'
                   YGrid: 'off'
                  YLabel: [1x1 Text]
                    YLim: [-1 1]
                YLimMode: 'auto'
              YMinorGrid: 'off'
              YMinorTick: 'off'
                  YScale: 'linear'
                   YTick: [-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1]
              YTickLabel: {11x1 cell}
          YTickLabelMode: 'auto'
      YTickLabelRotation: 0
               YTickMode: 'auto'
                  ZColor: [0.1500 0.1500 0.1500]
              ZColorMode: 'auto'
```

[13] Output of `get(gca)` (Continued). #

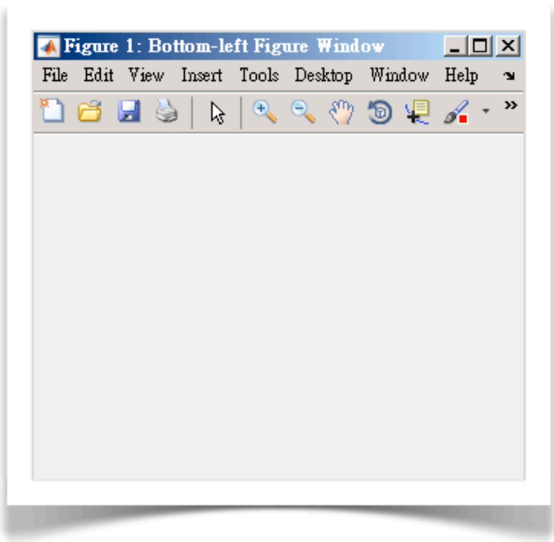
```
                    ZDir: 'normal'
                   ZGrid: 'off'
                  ZLabel: [1x1 Text]
                    ZLim: [-1 1]
                ZLimMode: 'auto'
              ZMinorGrid: 'off'
              ZMinorTick: 'off'
                  ZScale: 'linear'
                   ZTick: [-1 0 1]
              ZTickLabel: ''
          ZTickLabelMode: 'auto'
      ZTickLabelRotation: 0
               ZTickMode: 'auto'
```

5.3 Figure Objects

Example05_03.m: Figures

[1] The following commands demonstrate some important properties of a **Figure** object. Carefully observe the outcome of each command.

```
clear
scrsz = get(groot, 'ScreenSize');
h1 = figure;
    h1.Position = [20, 60, scrsz(3)/3, scrsz(4)/2];
    h1.Name = 'Bottom-left Figure Window';
h2 = figure;
    h2.Visible = 'off';
    h2.Units = 'normalized';
    h2.Position = [0.1, 0.2, 0.3, 0.4];
    h2.Visible = 'on';
    h2.Color = [0.8, 0.8, 0.8];
    h2.Name = 'A Window of Gray Background';
    h2.NumberTitle = 'off';
    h2.ToolBar = 'none';
    h2.MenuBar = 'none';
delete(h1)
delete(h2)
```



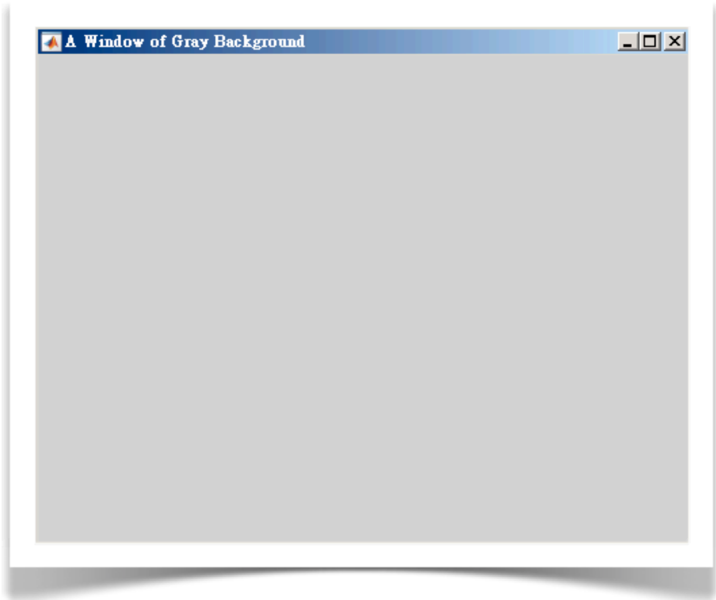


Table 5.3 Figure Properties

Properties	Description
`Color`	Background color
`MenuBar`	Menu bar display
`ToolBar`	Toolbar display
`Name`	Title
`NumberTitle`	Display of title number
`Position`	Position and size of drawable area
`Units`	Units of measurement (pixels)
`Visible`	Figure window visibility
`Resize`	Resize mode (on)

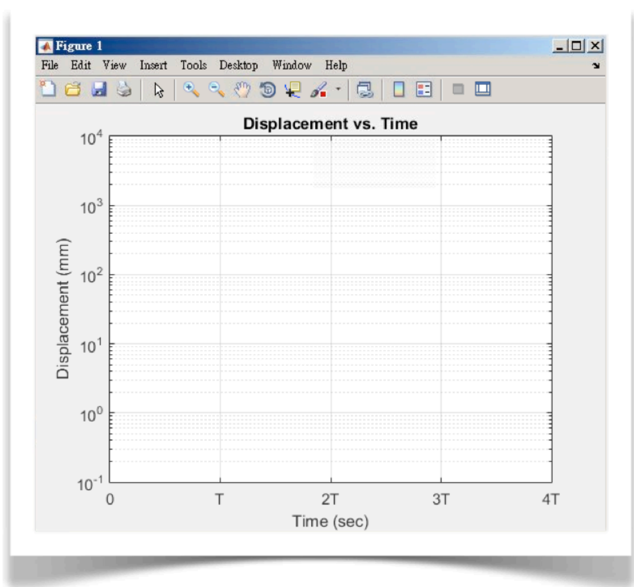
Details and More:
Help>MATLAB>Graphics> Graphics Objects>Graphics Object Properties>Top-Level Object>Figure Properties

5.4 Axes Objects

Example05_04a.m: Axes Properties

[1] An **Axes** object includes not only the *x*-axis and *y*-axis and their tick marks and labels, but also everything drawn on the **Axes**. The following commands demonstrate some important properties of **Axes** objects. Observe the outcome of each command.

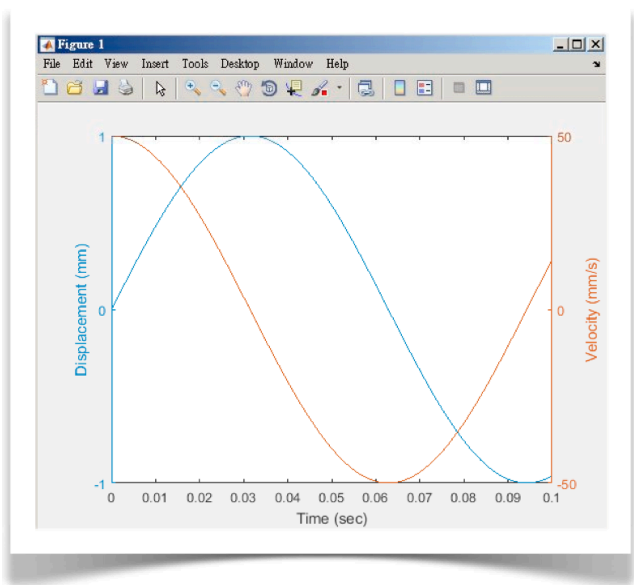
```
clear
h = axes;
    xlabel('Time (sec)');
    ylabel('Displacement (mm)');
    title('Displacement vs. Time');
    axis([0, 20, 0, 10000]);
    grid on
    box on
    h.YScale = 'log';
    h.XTick = [0, 5, 10, 15, 20];
    h.XTickLabel = {'0','T','2T','3T','4T'};
    h.FontSize = 11;
delete(h)
delete(gcf)
```



Example05_04b.m: Overlapping Axes

[4] The following commands demonstrate multiple axes in a figure.

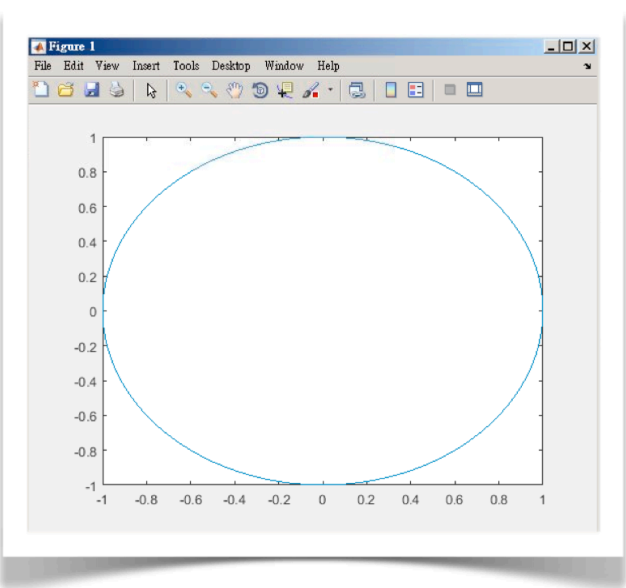
```
clear
t = linspace(0,0.1); w = 50;
y = sin(w*t);
v = w*cos(w*t);
yyaxis left
hLine1 = plot(t, y);
xlabel('Time (sec)')
ylabel('Displacement (mm)')
yyaxis right
hLine2 = plot(t, v);
ylabel('Velocity (mm/s)')
delete(hLine1)
delete(hLine2)
delete(gca)
delete(gcf)
```

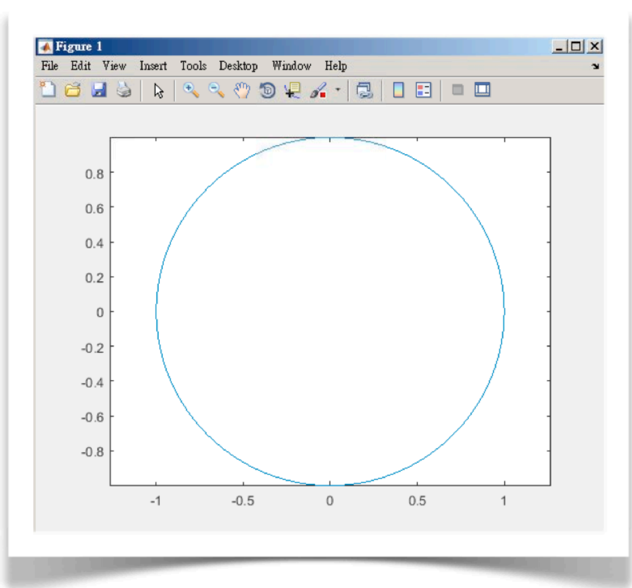


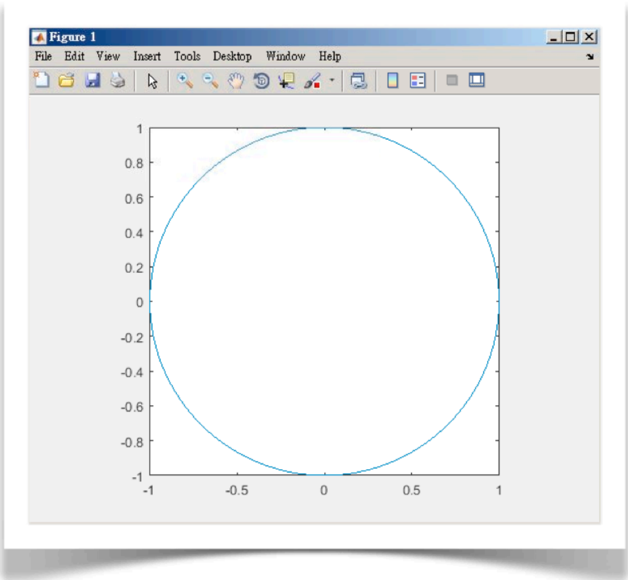
Example05_04c.m: Axes Scaling

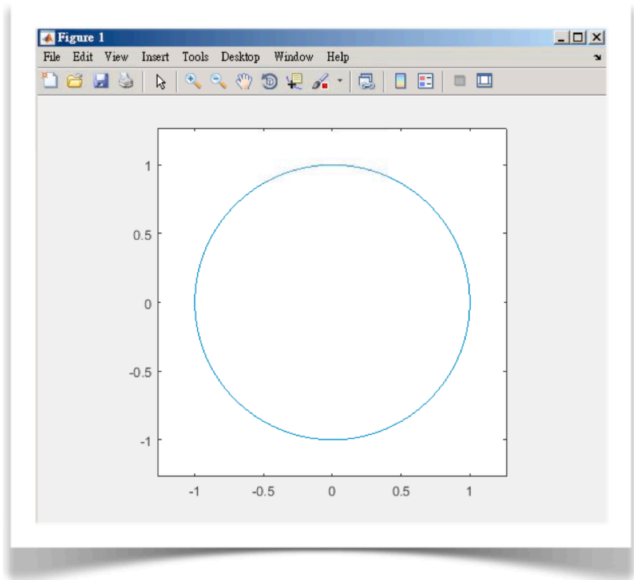
[7] The following commands demonstrate some additional **Axes** options.

```
clear
t = linspace(0,2*pi);
plot(cos(t), sin(t))
axis equal
limits = axis;
axis square
axis([limits(1),limits(2),limits(1), limits(2)])
delete(gca)
delete(gcf)
```









Example05_04d.m: Subplots

[13] The following commands demonstrate **subplots** in a figure. →

```
clear
t = linspace(0,0.1); w = 50;
y = sin(w*t);
v = w*cos(w*t);
a = -w*w*sin(w*t);
h1 = subplot(2,2,1);
plot(t,y), xlabel('Time'), ylabel('Displacement')
h2 = subplot(2,2,2);
plot(t,v), xlabel('Time'), ylabel('Velocity')
h3 = subplot(2,2,3);
plot(t,a), xlabel('Time'), ylabel('Acceleration')
h4 = subplot(2,2,4);
plot(y,a), xlabel('Displacement'), ylabel('Acceleration')
delete(h1)
delete(h2)
delete(h3)
delete(h4)
delete(gcf)
```

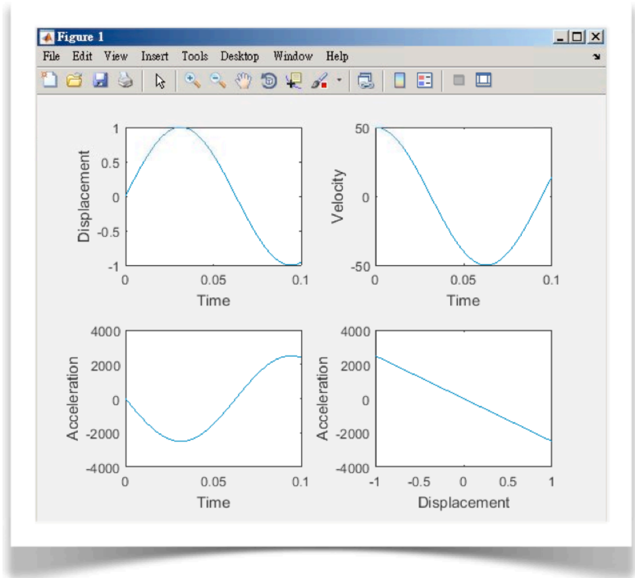
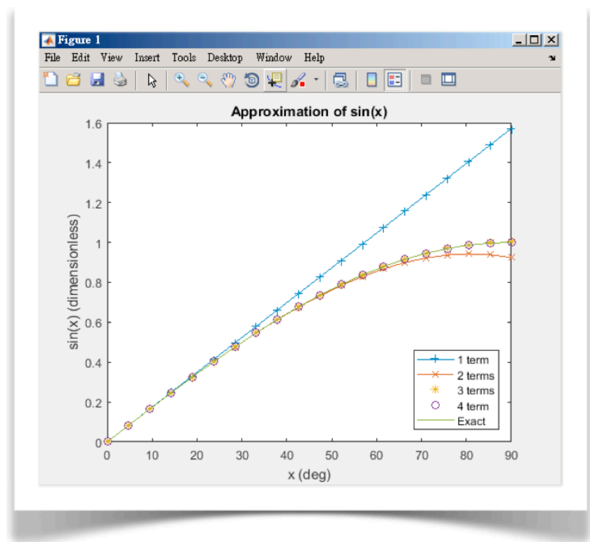


Figure 1
File Edit View Insert Tools Desktop Window Help
Displacement
Velocity
Time
Acceleration
Time
Acceleration
Displacement

Table 5.4a Axes Functions	
Functions	Description
`ha = axes`	Create axes graphics object
`axis(limits)`	Set axis limits
`axis equal`	Use the same length for the data units along each axis
`axis square`	Use the same length for the axis lines
`box on`	Display axes box outline
`ha = gca`	Get current axes
`grid on`	Display of grid lines
`title(text)`	Title for axes
`xlabel(text)`	Label x-axis
`ylabel(text)`	Label y-axis
`yyaxis right`	Specify the active side for the *y*-axis

Table 5.4b Axes Properties	
Properties	Description
`FontSize`	Font size (10 points)
`Position`	Position and size of axes
`Units`	Units of measurement (normalized)
`XLim, YLim, ZLim`	Minimum and maximum axis limits
`XScale, YScale, ZScale`	Scale of values along axis
`XTick, YTick, ZTick`	Tick mark locations
`XTickLabel, YTickLabel, ZTickLabel`	Tick mark labels
Details and More: Help>MATLAB>Graphics>Graphics Objects>Graphics Object Properties>Axes Properties	

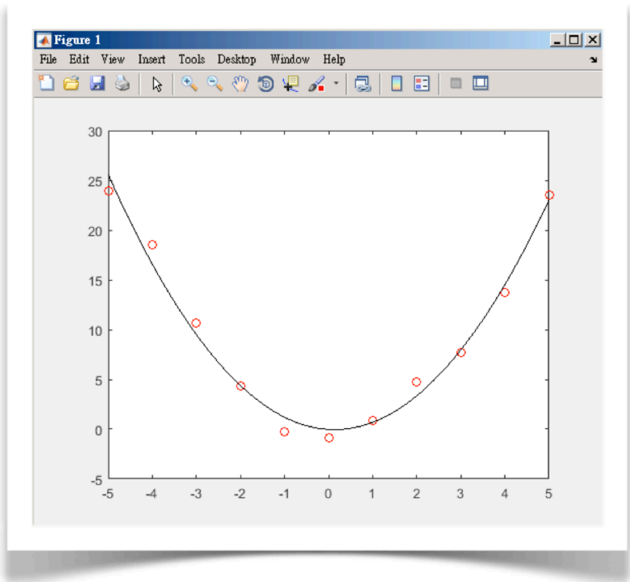
5.5 Line Objects



Example05_05a.m

[2] The following commands demonstrate the use of line styles, colors, and marker types. →

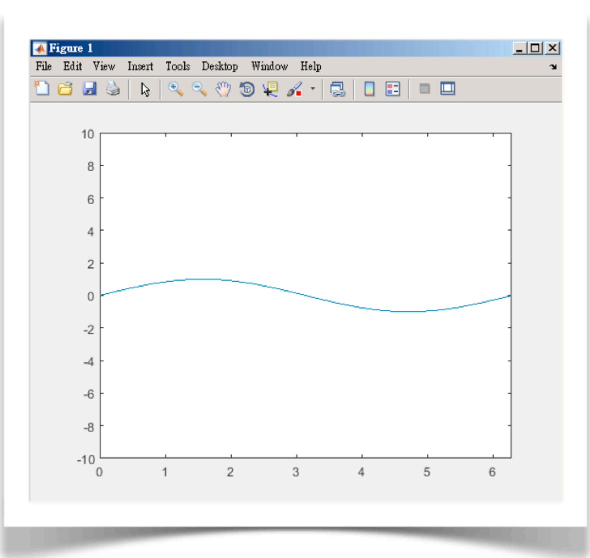
```
clear
x1 = [-5 -4 -3 -2 -1 0 1 2 3 4 5];
y1 = [23.9, 18.5, 10.7, 4.31, -0.26, -0.87, 0.82, 4.79, 7.67, 13.7, 23.5];
p = polyfit(x1, y1, 2)
x2 = linspace(-5,5);
y2 = polyval(p, x2);
h = plot(x1, y1, 'or', x2, y2, '-k');
delete(h(1))
delete(h(2))
delete(gca)
delete(gcf)
```



Example05_05b.m

[5] The following commands demonstrate some additional line properties.

```
clear
x = linspace(0,2*pi);
y = sin(x);
h = plot(x, y);
axis([0, 2*pi, -10, 10])
h.YData = 5*sin(x);
h.YData = 10*sin(x);
```



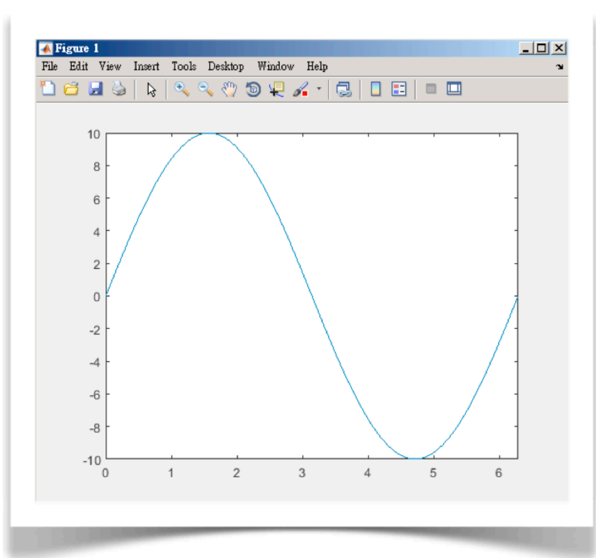


Table 5.5a Line Styles, Colors, and Marker Types

Line style		Color		Marker Type			
Symbol	Description	Symbol	Description	Symbol	Description	Symbol	Description
`-`	Solid	`r`	Red	`+`	Plus sign	`^`	Up triangle
`--`	Dashed	`g`	Green	`o`	Circle	`v`	Down triangle
`:`	Dotted	`b`	Blue	`*`	Asterisk	`>`	Right triangle
`-.`	Dashed-dot	`c`	Cyan	`.`	Point	`<`	Left triangle
		`m`	Magenta	`x`	Cross	`p`	Pentagram
		`y`	Yellow	`s`	Square	`h`	hexagram
		`k`	Black	`d`	Diamond		
		`w`	White				

Details and More: Help>MATLAB>Graphics>2-D and 3-D Plots>Line Plots>LineSpec (Line Specification)

Table 5.5b Line Functions (2-D)

Functions	Description
`plot(x,y,lineSpec)`	2-D line plot
`loglog(x,y,lineSpec)`	Log-log scale plot
`semilogx(x,y,lineSpec)`	Semilogarithmic plot
`semilogy(x,y,lineSpec)`	Semilogarithmic plot
`fplot(fun,linsSpec)`	Plot expression or function (2-D)
`fimplicit(fun,lineSpec)`	Plot 2-D implicit function

Details and More: Help>MATLAB>Graphics>2-D and 3-D Plots>Line Plots

Table 5.5c Line Properties

Properties	Description
Color	Line color
LineStyle	Line style
LineWidth	Line width (default 0.5)
Marker	Marker symbol
MarkerEdgeColor	Marker outline color
MarkerFaceColor	Marker fill color
MarkerSize	Marker size (default 6)
XData	x values
YData	y values
ZData	z values

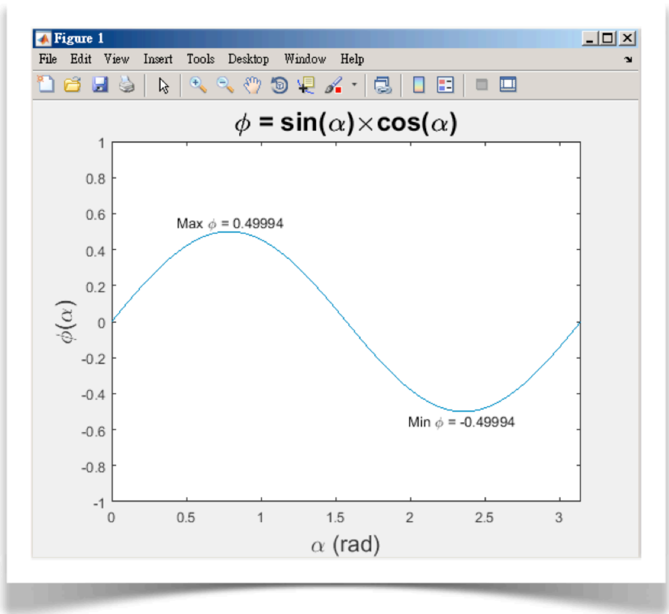
Details and More
Help>MATLAB>Graphics>Graphics Objects>Graphics Object Properties>Chart Objects>Chart Line Properties

5.6 Text Objects

Example05_06a.m

[1] The following commands demonstrate some important properties of **Text** objects. Observe the outcome of each command.

```
clear
alpha = linspace(0, pi);
phi = sin(alpha).*cos(alpha);
plot(alpha, phi)
axis([0, pi, -1, 1])
hx = xlabel('\alpha (rad)');
hy = ylabel('\phi(\alpha)');
ht = title('\phi = sin(\alpha)\timescos(\alpha)');
    hx.FontSize = 16;
    hy.FontSize = 16;
    ht.FontSize = 18;
[value, index] = max(phi);
hmax = text(alpha(index), value, ['Max \phi = ', num2str(value)]);
    hmax.HorizontalAlignment = 'center';
    hmax.VerticalAlignment = 'bottom';
[value, index] = min(phi);
hmin = text(alpha(index), value, ['Min \phi = ', num2str(value)]);
    hmin.HorizontalAlignment = 'center';
    hmin.VerticalAlignment = 'top';
delete(hx)
delete(hy)
delete(ht)
delete(hmax)
delete(hmin)
delete(gcf)
```



Example05_06b.m

[4] The following commands demonstrate a more sophisticated way of using function `text`. The dimmed lines (lines 26-32) are copied from Example05_05a.m (page 224).

```
26  clear
27  x1 = [-5 -4 -3 -2 -1 0 1 2 3 4 5];
28  y1 = [23.9, 18.5, 10.7, 4.31, -0.26, -0.87, 0.82, 4.79, 7.67, 13.7, 23.5];
29  p = polyfit(x1, y1, 2)
30  x2 = linspace(-5,5);
31  y2 = polyval(p, x2);
32  h = plot(x1, y1, 'or', x2, y2, '-k');
33  for k = 1:length(x1)
34      txt{k} = sprintf('(%g,%g)', x1(k), y1(k));
35  end
36  text(x1, y1-0.5, txt, ...
37      'HorizontalAlignment', 'center', 'VerticalAlignment', 'top')
```

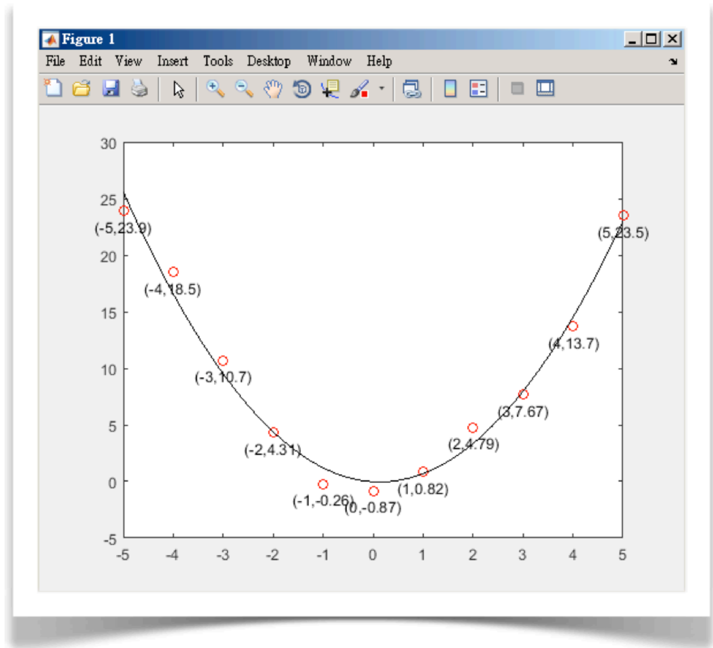


Table 5.6a Geek Letters and Math Symbols

Symbol	Syntax	Symbol	Syntax	Symbol	Syntax
α	`\alpha`	σ	`\sigma`	$\times$	`\times`
β	`\beta`	τ	`\tau`	$\div$	`\div`
γ	`\gamma`	ϕ	`\phi`	$\circ$	`\circ`
δ	`\delta`	χ	`\chi`	$\surd$	`\surd`
ε	`\epsilon`	ψ	`\psi`	$\rightarrow$	`\rightarrow`
ζ	`\zeta`	ω	`\omega`	$\leftarrow$	`\leftarrow`
η	`\eta`	Γ	`\Gamma`	$\uparrow$	`\uparrow`
θ	`\theta`	Δ	`\Delta`	$\downarrow$	`\downarrow`
λ	`\lamda`	Π	`\Pi`	(bold face)	`\bf`
μ	`\mu`	Σ	`\Sigma`	(italic)	`\it`
ν	`\nu`	Φ	`\Phi`	(remove)	`\rm`
ξ	`\xi`	Ψ	`\Psi`	(superscript)	`^`
π	`\pi`	Ω	`\Omega`	(subscript)	`_`
ρ	`\rho`				

Details and More: Help>MATLAB>Graphics>Graphics Objects>Graphics Object Properties> Primitive Objects>Text Properties>Interpreter

Table 5.6b Text Functions

Functions	Description
`text(x,y,text)`	Create axes graphics object
`title(text)`	Title for axes
`xlabel(text)`	Label x-axis
`ylabel(text)`	Label y-axis

Details and More: Help>MATLAB>Graphics>Formatting and Annotation>Titles and Labels

Table 5.6c Text Properties

Properties	Description
Color	Text color (black)
FontName	Font name (Helvetica)
FontSize	Font size (10 points)
HorizontalAlignment	Horizontal alignment (left)
Position	Position of text
String	Text to display
Units	Position and extent units (data)
VerticalAlignment	Vertical alignment (middle)

Details and More:
Help>MATLAB>Graphics>Graphics Objects>Graphics Object Properties>Primitive Objects>Text Properties

5.7 Legend Objects

Example05_07.m: Legends

[1] We've shown in Example02_12e.m (page 112) that, when a **Figure** contains multiple curves, adding a **Legend** improves the readability of the **Figure**. The following commands demonstrate the properties of **Legend** objects. The dimmed lines (lines 1-14) are duplicated from Example02_12e.m.

```
 1  clear
 2  x = linspace(0,pi/2,20);
 3  n = 4;
 4  k = (1:n);
 5  [X, K] = meshgrid(x, k);
 6  sinx =cumsum(((-1).^(K-1)).*(X .^ (2*K-1))./factorial(2*K-1));
 7  plot(x*180/pi, sinx(1,:), '+-', ...
 8       x*180/pi, sinx(2,:), 'x-', ...
 9       x*180/pi, sinx(3,:), '*', ...
10       x*180/pi, sinx(4,:), 'o', ...
11       x*180/pi, sin(x))
12  title('Approximation of sin(x)')
13  xlabel('x (deg)')
14  ylabel('sin(x) (dimensionless)')
15  h = legend('1 term', '2 terms', '3 terms', '4 terms', 'Exact');
16      h.Position = [0.6, 0.2, 0.25, 0.2];
17      h.FontSize = 16;
18      h.String{5} = 'sin(x)';
19      h.Box = 'off';
```

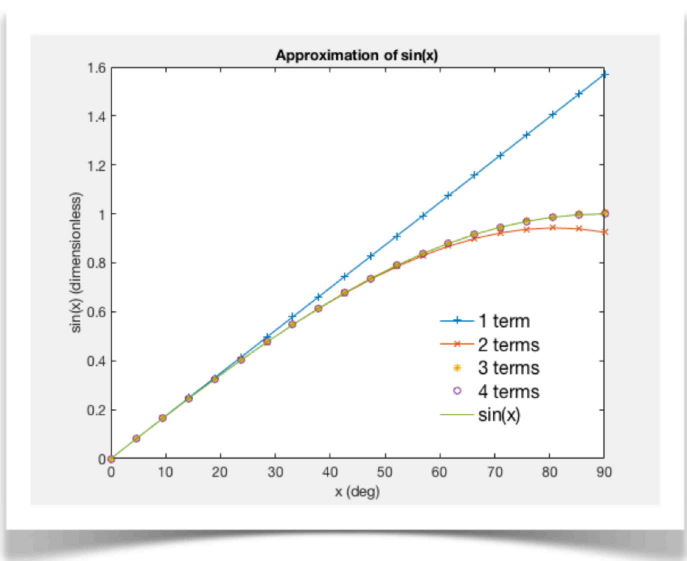


Table 5.7a Legend Functions	
Functions	Description
`legend(labels,name,value)`	Add legend to graph
Details and More: `>> doc legend`	

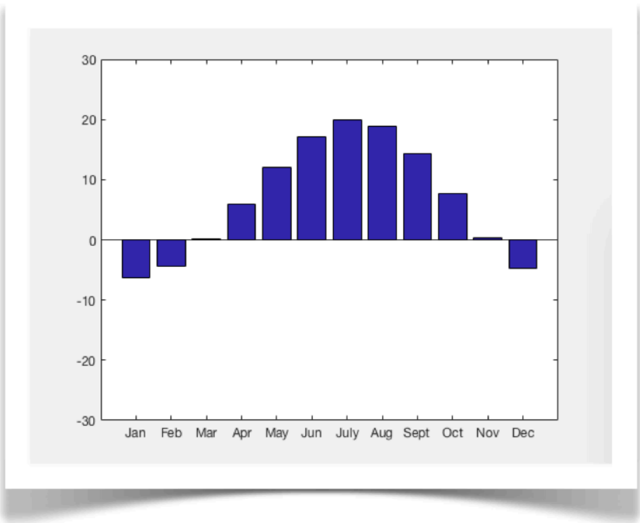
Table 5.7b Legend Properties	
Properties	Description
Box	Box outline (on)
FontSize	Font size (9 points)
Location	Location of legend (northeast)
Position	Custom position and size
String	Legend entry description
Units	Position units (normalized)
Details and More: *Help>MATLAB>Graphics>Graphics Objects>Graphics Object Properties>Illustration Objects>Legend Properties*	

5.8 Bar Plots

Example05_08.m: Bar Plots

[1] The following commands demonstrate the creation of bar plots. Observe the outcome of each command.

```
clear
USA = [-6.3, -4.3, 0.1, 5.9, 12.1, 17.1, ...
       19.9, 18.9, 14.4, 7.7, 0.4, -4.8];
hb = bar(USA);
ha = gca;
    axis([0, 13, -30, 30])
    ha.XTickLabel = {'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', ...
                     'July', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec'};
CAN = [-24.6, -23.3, -18.7, -9.8, -0.3, 7.2, ...
       11.1, 9.5, 3.5, -4.4, -14.5, -21.5];
GBR = [3.0, 3.0, 4.7, 6.7, 9.8, 12.8, ...
       14.4, 14.3, 12.2, 9.5, 5.5, 3.9];
y = [USA', CAN', GBR'];
delete(hb);
hold on
hb = bar(y);
    hb(1).BarWidth = 1.0;
    hb(2).BarWidth = 1.0;
    hb(3).BarWidth = 1.0;
title('Average Temperature (1961-1999)')
ylabel('Temperature (\circC)')
legend('United States', 'Canada', 'United Kingdom', 'Location', 'best')
```



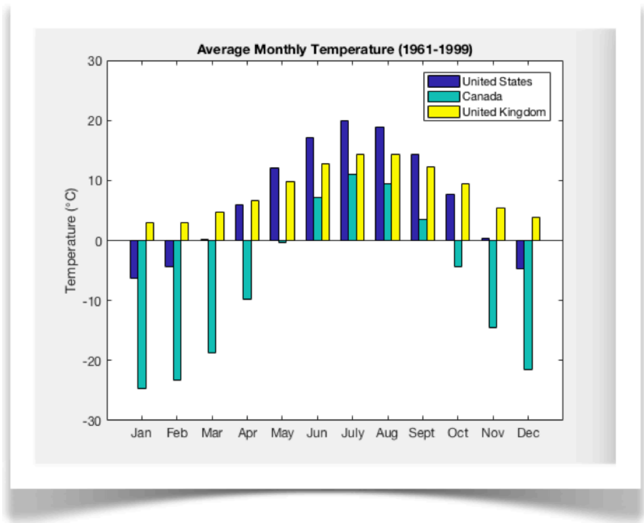


Table 5.8a Bar Functions	
Functions	Description
`bar(y,width)`	Plot bar graph
`bar(y,name,value)`	Plot bar graph
`barh(y,width)`	Plot horizontal bar graph
`bar3(y,width)`	Plot 3-D bar graph
`bar3h(y,width)`	Plot horizontal 3-D bar graph
Details and More: Help>MATLAB>Graphics>2-D and 3-D Plots>Pie Charts, Bar Plots, and Histograms	

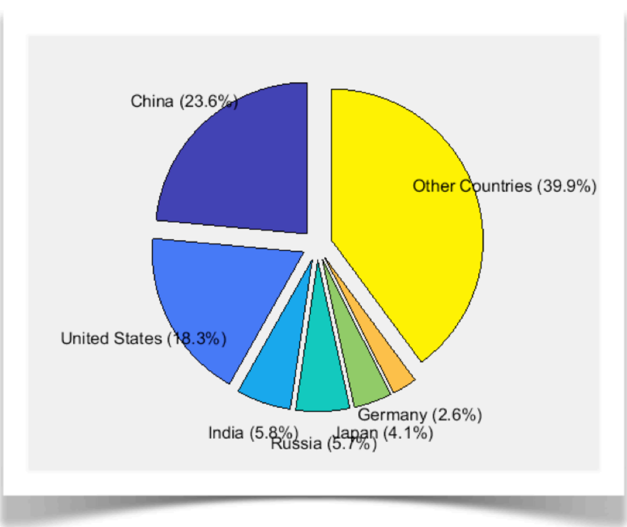
Table 5.8b Bar Properties	
Properties	Description
BarWidth	Relative width of bars (0.8)
BaseValue	Baseline location
EdgeColor	Bar outline color
FaceColor	Bar fill color
XData	Bar locations
YData	Bar length
Details and More: Help>MATLAB>Graphics>Graphics Objects>Graphics Object Properties>Chart Objects>Bar Properties	

5.9 Pie Plots

Example05_09.m: Pie Plots

[1] It is estimated that, in 2008, the total CO_2 emission in the world was 29.85 Bt (billion tons), of which, in descending order, 7.03 Bt was produced by China, 5.46 Bt by United States, 1.74 Bt by India, 1.71 Bt by Russia, 1.21 Bt by Japan, and 0.79 Bt by Germany (*Data source: http://data.worldbank.org/data-catalog/climate-change*). The following script generates a **Pie** plot [2] that shows these data.

```
clear
world = 29.85;
CHN = 7.03; USA = 5.46; IND = 1.74;
RUS = 1.71; JPN = 1.21; DEU = 0.79;
others = world - CHN - USA - IND - RUS - JPN - DEU;
x = [CHN, USA, IND, RUS, JPN, DEU, others];
explode = [1, 1, 1, 1, 1, 1, 1];
countries = {'China', 'United States', 'India', ...
    'Russia', 'Japan', 'Germany', 'Other Countries'};
for k = 1:7
    labels{k} = [countries{k}, sprintf(' (%.1f%%)', x(k)/world*100)];
end
h = pie(x, explode, labels)
    h(2).FontSize = 12;
    h(4).FontSize = 12;
    h(6).FontSize = 12;
    h(8).FontSize = 12;
    h(10).FontSize = 12;
    h(12).FontSize = 12;
    h(14).FontSize = 12;
```



```
h =
  1×14 graphics array:

  Columns 1 through 8
    Patch    Text     Patch    Text     Patch    Text     Patch    Text
  Columns 9 through 14
    Patch    Text     Patch    Text     Patch    Text
```

Table 5.9a Pie Functions	
Functions	Description
`pie(x, explode, labels)`	Pie chart
`pie3(x, explode, labels)`	3-D pie chart
Details and More: Help>MATLAB>Graphics>2-D and 3-D Plots>Pie Charts, Bar Plots, and Histograms	

Table 5.9b Patch Properties	
Properties	Description
FaceColor	Patch fill color
EdgeColor	Patch outline color
Details and More: Help>MATLAB>Graphics>Graphics Objects>Graphics Object Properties>Primitive Objects>Patch Properties	

5.10 3-D Line Plots

Example05_10.m: 3-D Line Plots

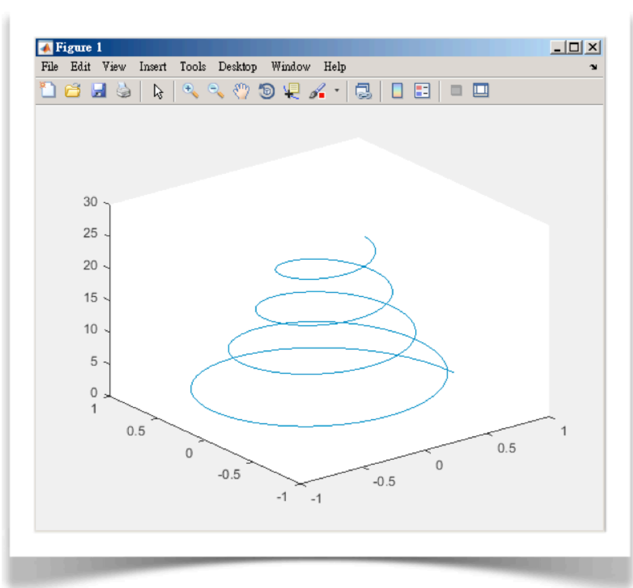
[1] The function `plot3(x,y,z)` plots a 3-D line by connecting the following points:

$$(x(k),\ y(k),\ z(k));\ k = 1,\ 2,\ \ldots$$

The following commands create a 3-D line plot (see [3-5], this and next pages) using the following parametric equations

$$x(t) = e^{-t/20}\cos t,\quad y(t) = e^{-t/20}\sin t,\quad z(t) = t$$

```
clear
z = linspace(0, 8*pi, 200);
x = exp(-z/20).*cos(z);
y = exp(-z/20).*sin(z);
plot3(x,y,z)
xlabel x, ylabel y, zlabel z
axis([-1, 1, -1, 1, 0, 8*pi])
h = gca; h.BoxStyle = 'full'; box on
grid on
axis vis3d
```



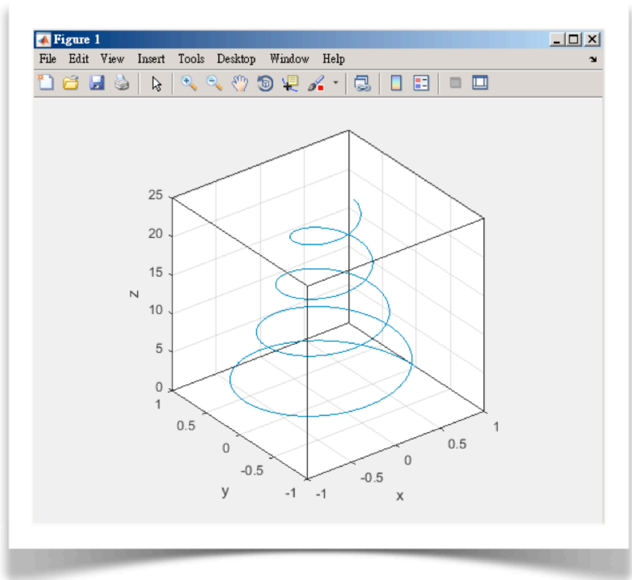


Table 5.10a 3-D Line Plot Functions	
Functions	Description
`plot3(x,y,z,lineSpec)`	Plot 3-D lines
`fplot3(funx,funy,funz,lineSpec)`	3-D parametric curve plotter
`fimplicit3(fun,lineSpec)`	Plot 3-D implicit function
`axis vis3d`	Freeze the aspect ratio properties.
Details and More: Help>MATLAB>Graphics>2-D and 3-D Plots>Line Plots	

Table 5.10b Additional Axes Properties	
Properties	Description
`BoxStyle`	Box style (back)
Details and More: Help>MATLAB>Graphics>Graphics Objects>Graphics Object Properties>Top-Level Objects>Axes Properties	

5.11 Surface and Mesh Plots

Example05_11a.m: Surface and Mesh Plots

[1] The function `surf(X,Y,Z)` generates a surface by connecting the points

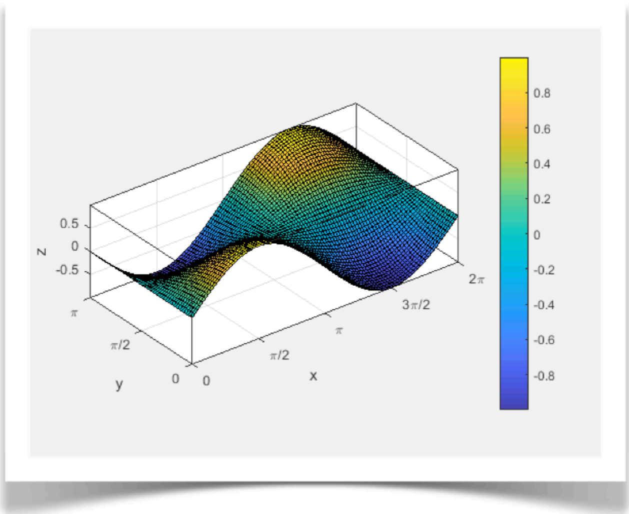
$$(X(i,j),\ Y(i,j),\ Z(i,j));\ i=1,\ 2,\ \ldots;\ j=1,\ 2,\ \ldots$$

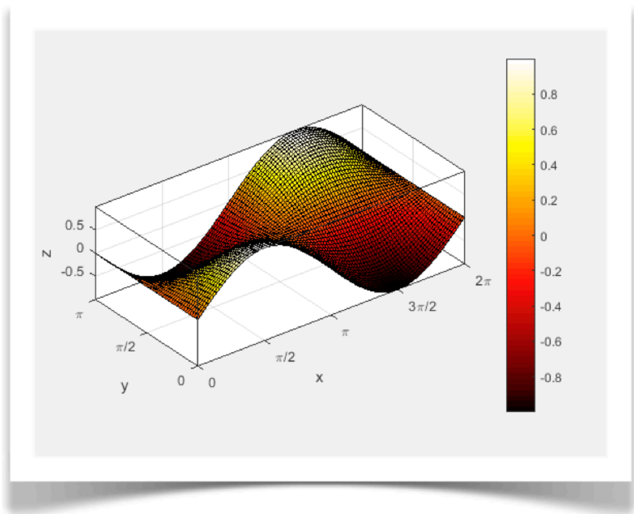
and then filling with faces between the edges. By default, each face is colored according to the current **colormap** (see Table 5.11a, page 243), and the edges are black-colored.

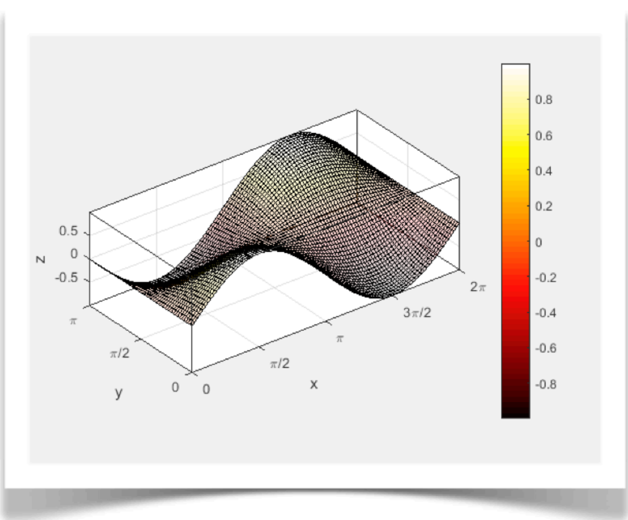
The following commands generate a surface described by

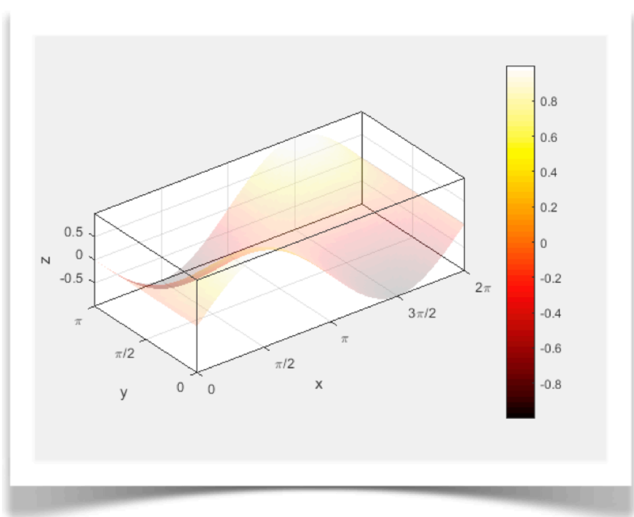
$$z(x,y)=\sin x\cdot\cos y \tag{a}$$

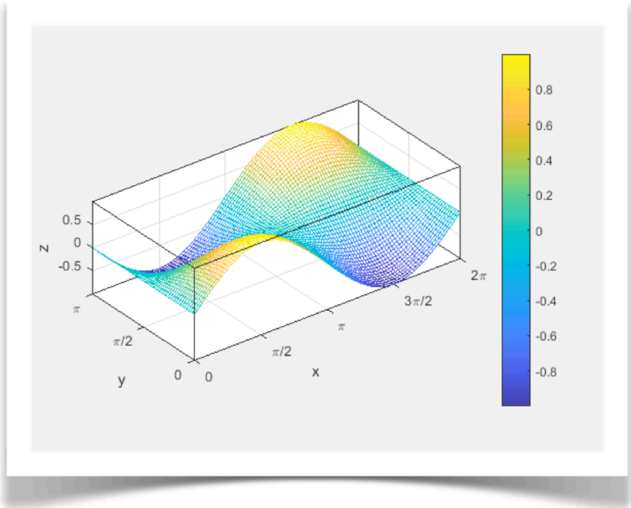
```
clear
x = linspace(0,2*pi,100);
y = linspace(0,  pi, 50);
[X,Y] = meshgrid(x, y);
Z = sin(X) .* cos(Y);
surf(X, Y, Z)
xlabel x, ylabel y, zlabel z
h = gca;
    axis([0, 2*pi, 0, pi, -1, 1])
    h.XTick = [0, pi/2, pi, 3*pi/2, 2*pi];
    h.YTick = [0, pi/2, pi];
    h.XTickLabel = {'0', '\pi/2', '\pi', '3\pi/2', '2\pi'};
    h.YTickLabel = {'0', '\pi/2', '\pi'};
    axis vis3d
    axis equal
    h.BoxStyle = 'full';
    box on
    grid on
colorbar
colormap hot
    hs.FaceAlpha = 0.2;
    hs.EdgeColor = 'none';
```

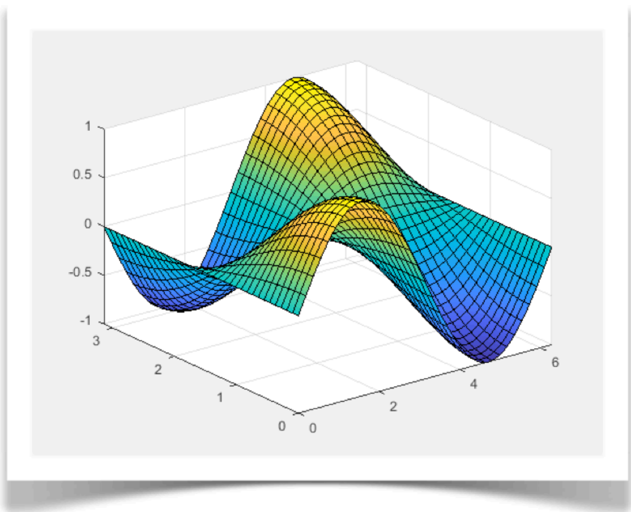

z
0.5
0
-0.5
y
π
π/2
0
x
0
π/2
π
3π/2
2π
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8


z
0.5
0
-0.5
y
π
π/2
0
x
0
π/2
π
3π/2
2π
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8


z
0.5
0
-0.5
y
π
π/2
0
x
0
π/2
π
3π/2
2π
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8


z
0.5
0
-0.5
y
π
π/2
0
x
0
π/2
π
3π/2
2π
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8


0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0.5
0
-0.5
z
π
π/2
y
0
0
π/2
π
3π/2
2π
x


1
0.5
0
-0.5
-1
3
2
1
0
0
2
4
6

Example05_11b.m

[9] Consider a spherical surface of radius $2a$ centered at the origin and a cylindrical surface of radius a and length $4a$, centered at $(a, 0, 0)$. The intersection of the two surfaces is a curve given by the following parametric equations:

$$x = a(1+\cos t), \quad y = a\sin t, \quad z = 2a\sin(t/2) \tag{b}$$

The following commands plot the two surfaces and the intersecting curve (see [11], next page).

```
clear
a = 1;
[X,Y,Z] = sphere;
surf(2*a*X,2*a*Y,2*a*Z)
hold on
[X,Y,Z] = cylinder;
surf(a*X+a,a*Y,4*a*Z-2*a)
shading interp
t = linspace(0,4*pi);
x = a*(1+cos(t));
y = a*sin(t);
z = 2*a*sin(t/2);
plot3(x,y,z)
axis equal
axis vis3d
xlabel x, ylabel y, zlabel z
view(45,30)
```

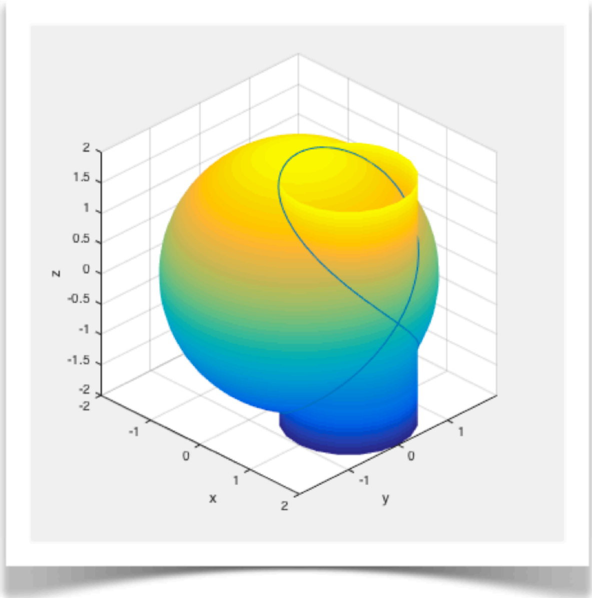


Table 5.11a Colormaps

Colormap Name	Color Scale
`parula`	
`jet`	
`hsv`	
`hot`	
`cool`	
`spring`	
`summer`	
`autumn`	
`winter`	
`gray`	
`bone`	
`copper`	
`pink`	
`lines`	
`colorcube`	
`prism`	
`flag`	
`white`	

Details and More: Type >> `doc colormap`

Table 5.11b Surface Functions

Functions	Description
`surf(X,Y,Z)`	Shaded surface plot
`mesh(X,Y,Z)`	Mesh plot
`fsurf(funx,funy,funz)`	Surface plotter
`colorbar`	Display color scale
`colormap map`	Set current colormap
`[X,Y,Z] = cylinder`	Generate cylinder
`[X,Y,Z] = sphere`	Generate sphere
`[X,Y,Z] = ellipsoid`	Generate ellipsoid
`[X,Y,Z] = peaks`	Generate a surface of "peaks"
`shading mode`	Set color shading properties
`view(az,el)`	View point specification

Details and More:
Help>MATLAB>Graphics>2-D and 3-D Plots>Surface, Volumes, and Polygons>Surface and Mesh Plots

Table 5.11c Surface Properties

Properties	Description
`EdgeColor`	Edge line color
`FaceAlpha`	Face transparency
`FaceColor`	Face color

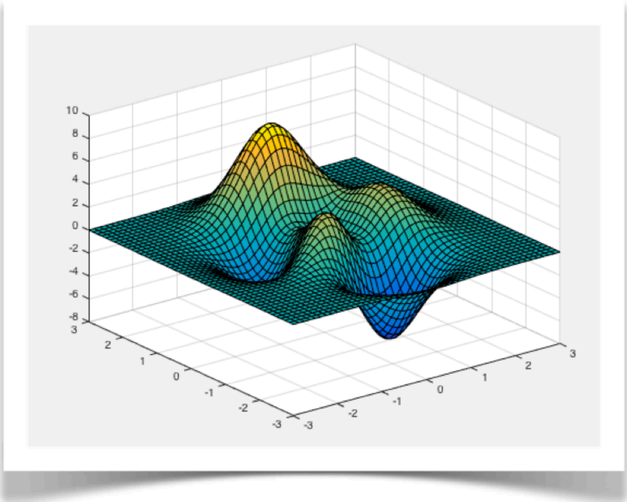
Details and More:
Help>MATLAB>Graphics>Graphics Objects>Graphics Object Properties>Chart Objects>Chart Surface Properties

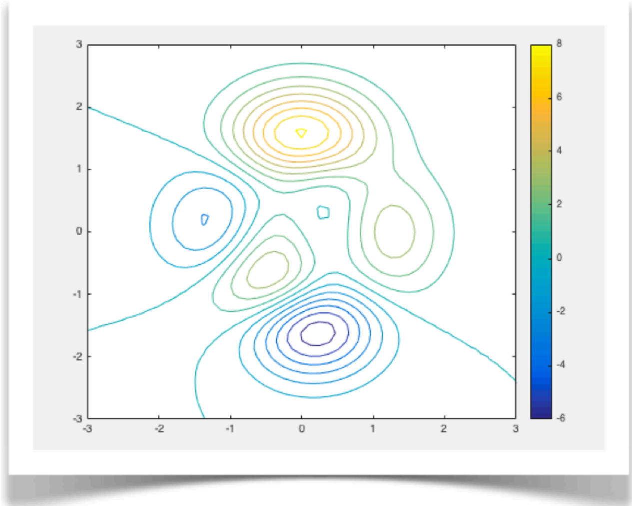
5.12 Contour Plots

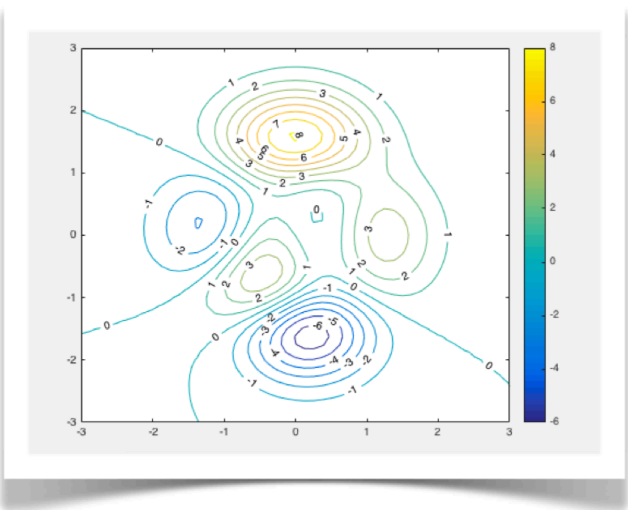
Example05_12.m: Contour Plots

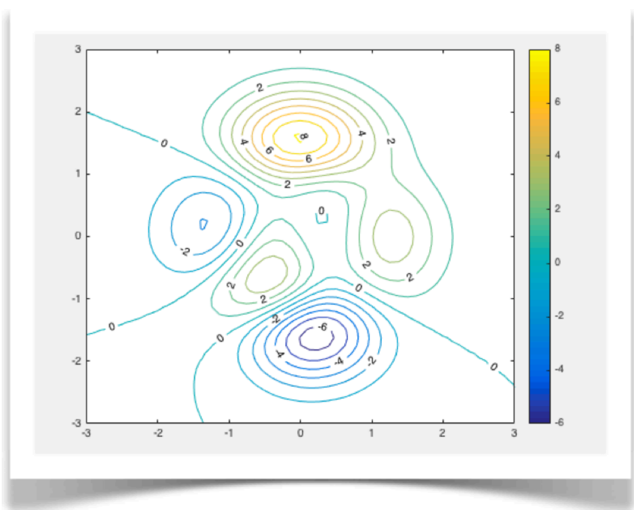
[1] The following commands generate a surface plot as shown in [3], next page, and its contour plots as shown in [3-8].

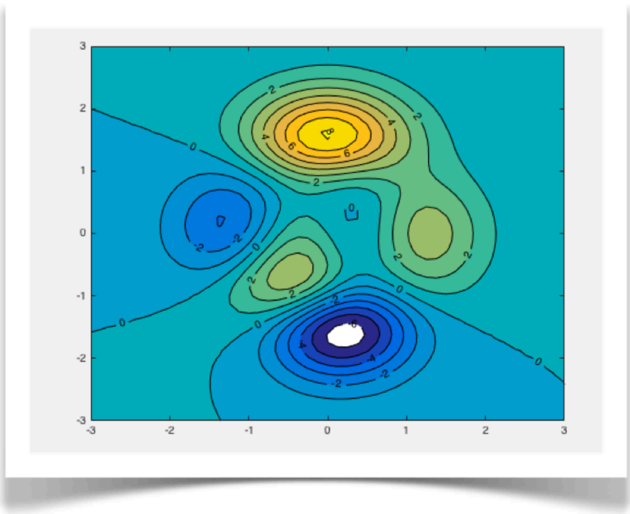
```
clear
[X,Y,Z] = peaks;
surf(X,Y,Z)
[C,h] = contour(X,Y,Z, [-6:8]);
colorbar
    h.ShowText = 'on';
    h.TextList = [-6:2:8];
[C,h] = contourf(X,Y,Z, [-6:8]);
clabel(C,h, [-6:2:8])
[C,h] = contour3(X,Y,Z, [-6:8]);
clabel(C,h, [-6:2:8])
```


10
8
6
4
2
0
-2
-4
-6
-8
3 2 1 0 -1 -2 -3
-3 -2 -1 0 1 2 3


3 2 1 0 -1 -2 -3
-3 -2 -1 0 1 2 3
8 6 4 2 0 -2 -4 -6


3 2 1 0 -1 -2 -3
-3 -2 -1 0 1 2 3
8 6 4 2 0 -2 -4 -6


3 2 1 0 -1 -2 -3
-3 -2 -1 0 1 2 3
8 6 4 2 0 -2 -4 -6



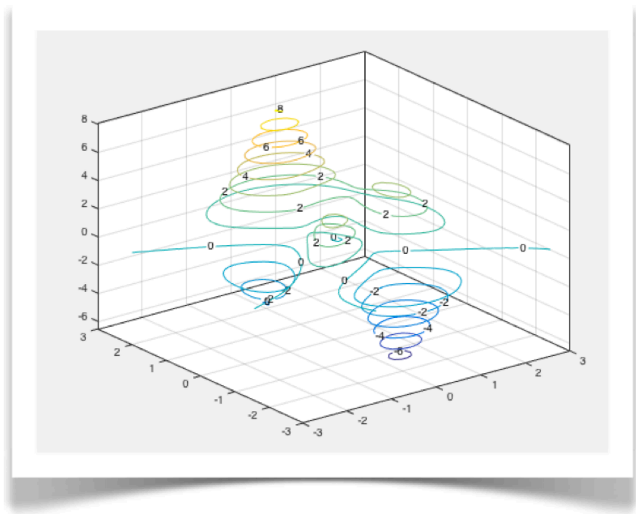


Table 5.12a Contour Functions	
Functions	Description
`[C,h] = contour(X,Y,Z,values)`	Contour plot
`[C,h] = contourf(X,Y,Z,values)`	Filled contour plot
`[c,h] = contour3(X,Y,Z,values)`	3-D contour plot
`clabel(C,h,values)`	Label contour plot
Details and More: Help>MATLAB>Graphics>2-D and 3-D Plots>Contour Plots	

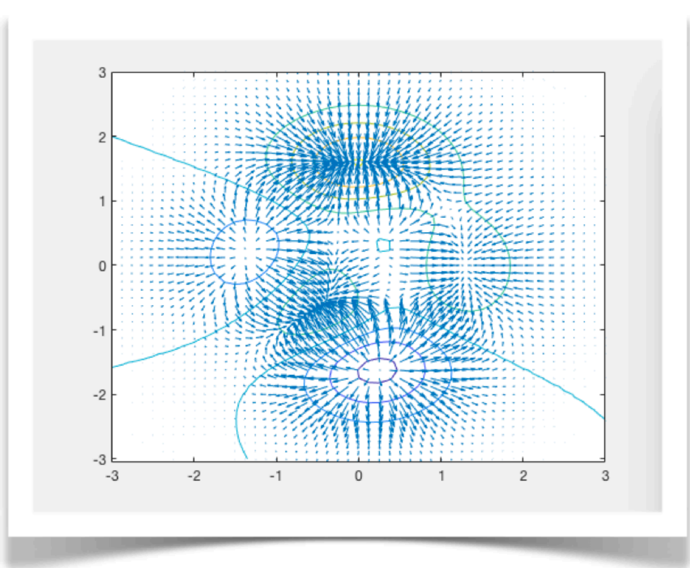
Table 5.12b Contour Properties	
Properties	Description
`ShowText`	Show contour line labels
`TextList`	Contour lines to label
Details and More: Help>MATLAB>Graphics>Graphics Objects>Graphics Object Properties>Chart Objects>Contour Properties	

5.13 Vector Plots

Example05_13a.m: 2-D Vector Plots

[1] The following commands generate a vector plot as shown in [3].

```
clear
[X,Y,Z] = peaks;
contour(X,Y,Z);
hold on
[U,V] = gradient(Z,0.2,0.2);
quiver(X,Y,U,V,3)
```



Example05_13b.m: 3-D Vector Plots

[4] The following commands generate a 3-D vector plot as shown in [6].

```
clear
[X,Y,Z] = peaks;
surf(X,Y,Z);
limits = axis
hold on
[U,V,W] = surfnorm(X,Y,Z);
quiver3(X,Y,Z,U,V,W)
axis(limits)
```

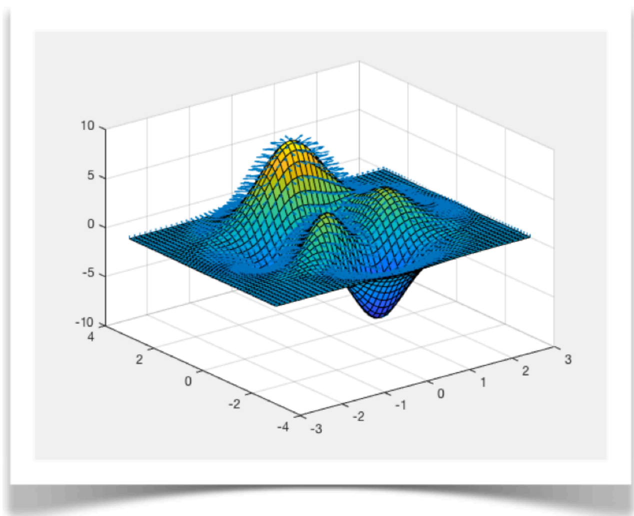


Table 5.13 Vector Plots Functions	
Functions	Description
`quiver(X,Y,U,V,scale)`	2-D Vector plot
`quiver3(X,Y,Z,U,V,W,scale)`	3-D vector plot
`[U,V] = gradient(Z)`	Compute gradient
`[U,V,W] = surfnorm(X,Y,Z)`	Compute surface normals
Details and More: Help>MATLAB>Graphics>2-D and 3-D Plots>Vector Fields	

5.14 Streamline Plots

Example05_14a.m: 2-D Streamlines

[1] The script below generates a streamline plot for the flow described by a velocity field (u, v),

$$u(x,y) = 0.3 + x$$

$$v(x,y) = 0.4 - y$$

```
clear
x = 0:0.1:1; y = 0:0.1:1;
[X,Y] = meshgrid(x,y);
U = 0.3+X; V = 0.4-Y;
quiver(X,Y,U,V)
sx = [0:0.1:1, zeros(1,11), 0:0.1:1];
sy = [zeros(1,11), 0:0.1:1, ones(1,11)];
SL = stream2(X,Y, U,V, sx,sy);
streamline(SL)
```

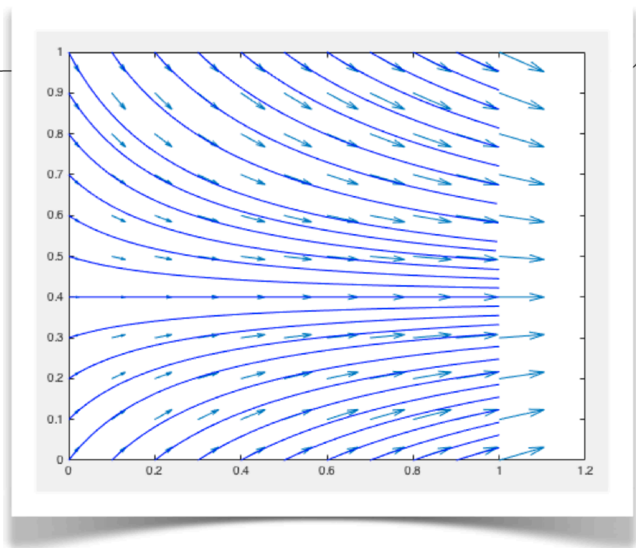


Table 5.14 Streamline Plots Functions	
Functions	Description
`streamline(SL)`	Streamline plot
`SL = stream2(X,Y, U,V, Sx,Sy)`	Calculate 2-D streamlines
`SL = stream3(X,Y,Z, U,V,W, Sx,Sy,Sz)`	Calculate 3-D streamlines
Details and More: Help>MATLAB>Graphics>2-D and 3-D Plots>Surfaces, Volumes, and Polygons> Volume Visualization>Vector Volume Data	

Example05_14b.m: 3-D Streamlines

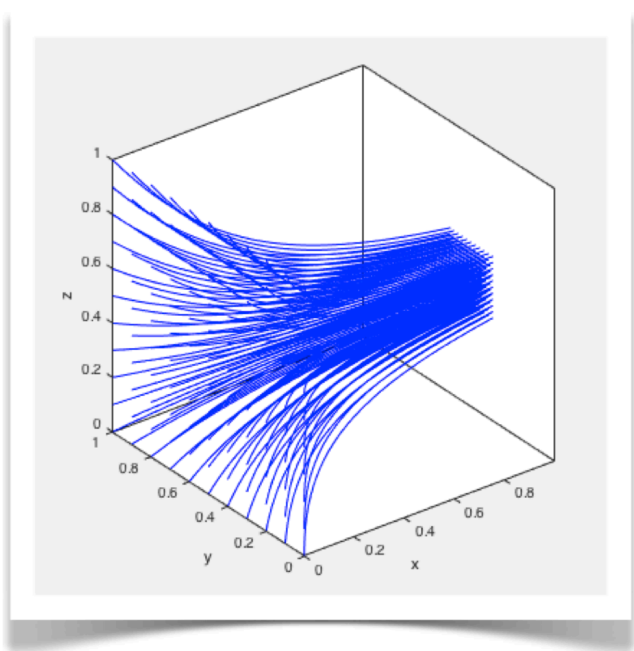
[4] The script below generates a 3-D streamline plot for the flow described by a 3-D velocity field (u, v, w),

$$u(x,y) = 0.3 + x$$

$$v(x,y) = 0.4 - y$$

$$w = 0.5 - z$$

```
clear
x = 0:0.1:1; y = 0:0.1:1; z = 0:0.1:1;
[X,Y,Z] = meshgrid(x,y,z);
U = 0.3+X; V = 0.4-Y; W = 0.5-Z;
% quiver3(X,Y,Z,U,V,W)
sx = 0;
sy = 0:0.1:1;
sz = 0:0.1:1;
[Sx, Sy, Sz] = meshgrid(sx,sy,sz);
SL = stream3(X,Y,Z, U,V,W, Sx,Sy,Sz);
streamline(SL)
view(3), axis vis3d, box on
xlabel('x'), ylabel('y'), zlabel('z')
```



5.15 Isosurface Plots

Example05_15.m: Isosurface Plots

[1] Imagine a potential function V in 3-D space,

$$V(x,y,z) = 0.2 + 0.3x + 0.4y + 0.5z + 0.5x^2 - 0.5y^2 - 0.5z^2$$

where $0 \le x \le 1$, $0 \le y \le 1$, and $0 \le z \le 1$. An isosurface is a surface on which a function has a specific common value. The following commands generate an isosurface plot for the potential function V.

```
clear
x = 0:0.05:1; y = 0:0.05:1; z = 0:0.05:1;
[X,Y,Z] = meshgrid(x,y,z);
V = 0.3*X+0.4*Y+0.5*Z+0.5*X.^2-0.5*Y.^2-0.5*Z.^2;
colorbar
hold on
for isovalue = 0.4:0.1:0.8
    isosurface(X,Y,Z,V, isovalue)
end
view(3), axis vis3d
xlabel('x'), ylabel('y'), zlabel('z')
```

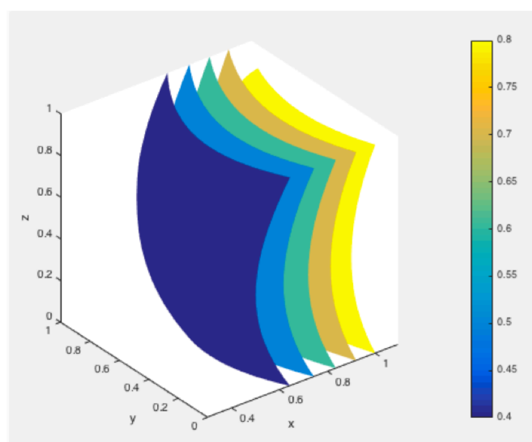
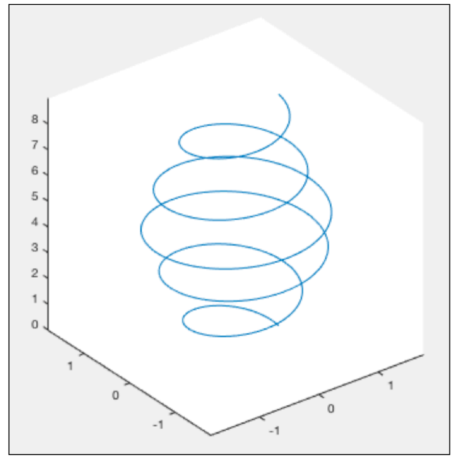
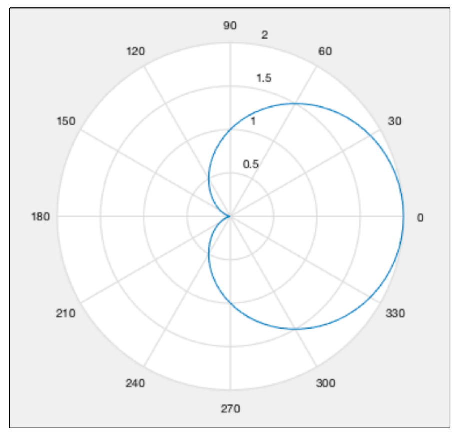
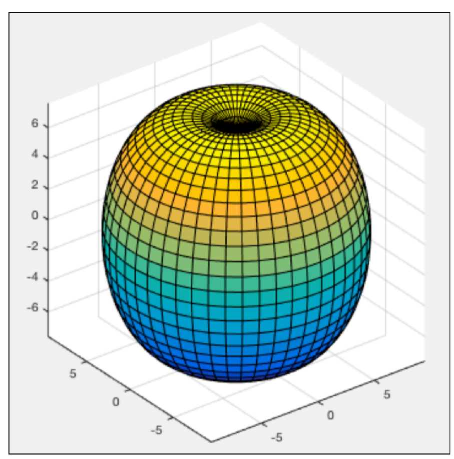


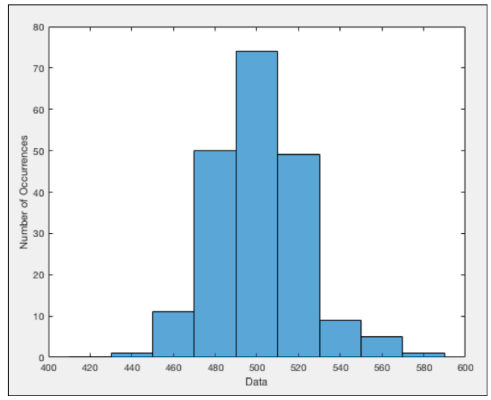
Table 5.15 Isosurface Plots Functions	
Functions	Description
`isosurface(X,Y,Z,V,isovalue)`	Isosurface plot
Details and More: Help>MATLAB>Graphics>2-D and 3-D Plots>Surfaces, Volumes, and Polygons> Volume Visualization>Scalar Volume Data	

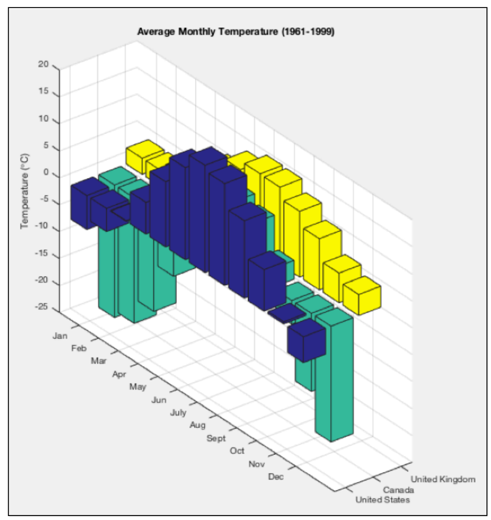
5.16 Additional Exercise Problems

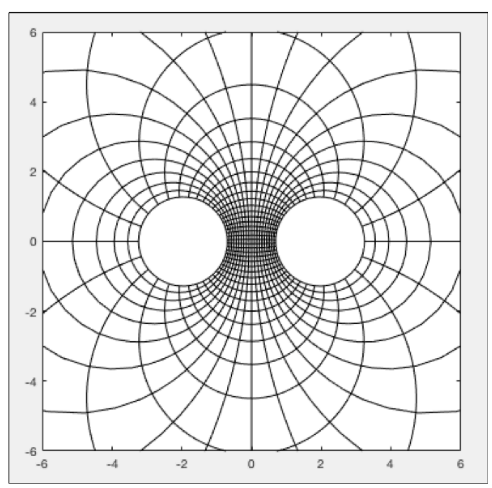


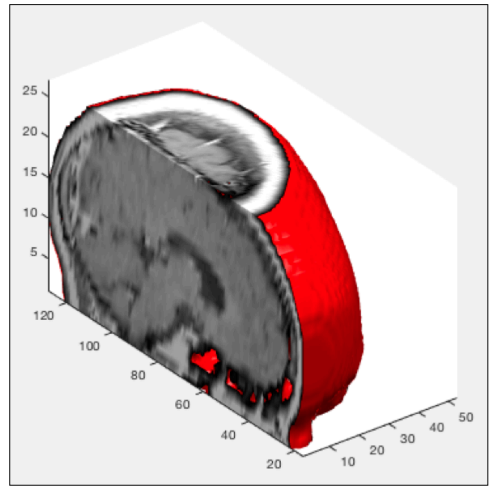





Number of Occurrences
Data
0
10
20
30
40
50
60
70
80
400
420
440
460
480
500
520
540
560
580
600


Average Monthly Temperature (1961-1999)
Temperature (°C)
20
15
10
5
0
-5
-10
-15
-20
-25
Jan
Feb
Mar
Apr
May
Jun
July
Aug
Sept
Oct
Nov
Dec
United Kingdom
Canada
United States


6
4
2
0
-2
-4
-6
-6
-4
-2
0
2
4
6


25
20
15
10
5
120
100
80
60
40
20
10
20
30
40
50

Chapter 6
Animations, Images, Audios, and Videos

Animation is often the best way than a static picture to present time-dependent data. An engineer often needs to deal with an image file, an audio file, or a video file. MATLAB provides many functions that can open and operate on various formats of image files, audio files, and video files.

6.1 Animation of Line Plots: Comet

Example06_01.m

[1] We've demonstrated an animation of a 2-D line plot in line 37, Example01_18.m, page 55. The following commands demonstrate an animation of a 3-D line plot, which was defined in 5.10[1], page 237.

```
clear
view(3)
axis([-1, 1, -1, 1, 0, 8*pi])
xlabel x, ylabel y, zlabel z
h = gca; h.BoxStyle = 'full'; box on
grid on
axis vis3d
hold on
z = linspace(0, 8*pi, 200);
x = exp(-z/20).*cos(z);
y = exp(-z/20).*sin(z);
comet3(x,y,z)
```

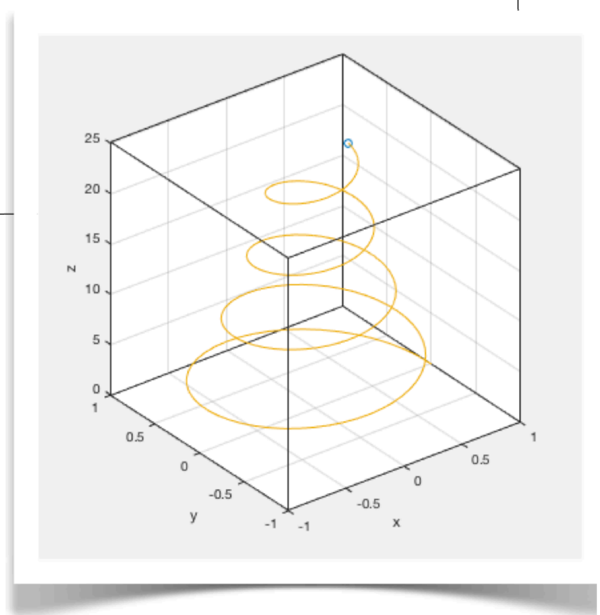


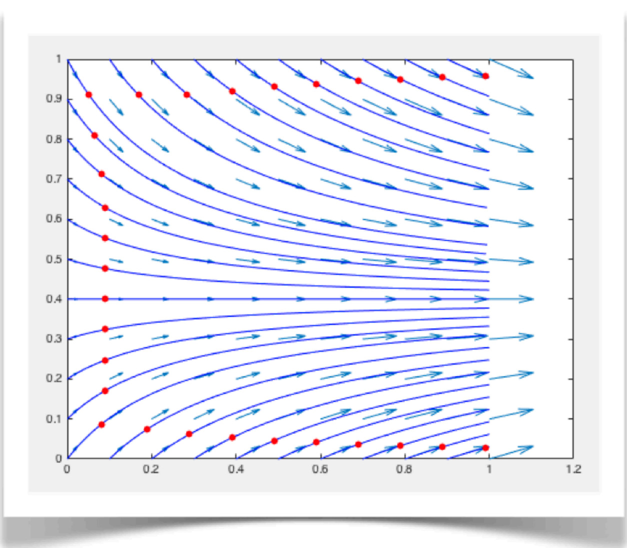
Table 6.1 Animated Line Plots Functions	
Functions	Description
`comet(x,y)`	Animate 2-D line plots.
`comet3(x,y,z)`	Animate 3-D line plots.
`animatedline(x,y)`	Create animated line (2-D)
`animatedline(x,y,z)`	Create animated line (3-D)
Details and More: Help>MATLAB>Graphics>2-D and 3-D Plots>Animation	

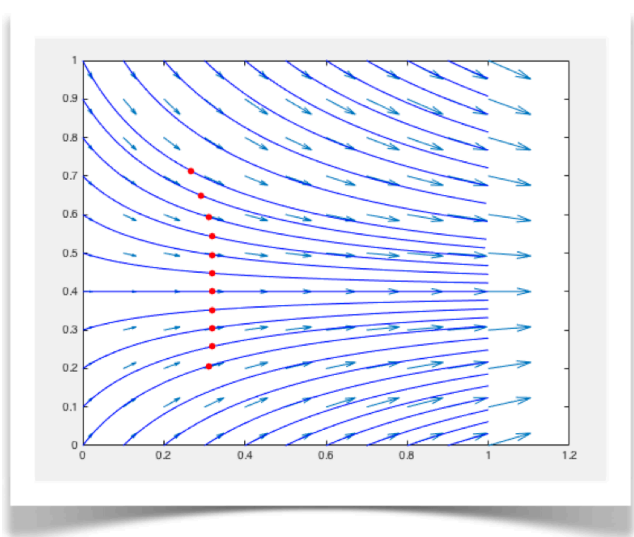
6.2 Stream Particles Animations

Example06_02a.m: Animation of 2-D Streamlines

[1] The following commands perform an animation of the flow described in 5.14[1], page 250. The dimmed lines (lines 1-9) are copied from Example05_14a.m, page 250.

```
1   clear
2   x = 0:0.1:1; y = 0:0.1:1;
3   [X,Y] = meshgrid(x,y);
4   U = 0.3+X; V = 0.4-Y;
5   quiver(X,Y,U,V)
6   sx = [0:0.1:1, zeros(1,11), 0:0.1:1];
7   sy = [zeros(1,11), 0:0.1:1, ones(1,11)];
8   SL = stream2(X,Y, U,V, sx,sy);
9   streamline(SL)
10  streamparticles(SL, ...
11      'Animate', 5, ...
12      'FrameRate', 30, ...
13      'ParticleAlignment', 'on')
14  sx = zeros(1,11);
15  sy = 0:0.1:1;
16  SL = stream2(X,Y, U, V, sx, sy);
17  streamparticles(SL, ...
18      'Animate', 5, ...
19      'FrameRate', 30, ...
20      'ParticleAlignment', 'on')
```





Example06_02b.m: Animation of 3-D Streamlines

[5] The following commands perform an animation of the flow described in 5.14[4], page 251. The dimmed lines (lines 21-33) are copied from Example05_14b.m, page 251.

```
21   clear
22   x = 0:0.1:1; y = 0:0.1:1; z = 0:0.1:1;
23   [X,Y,Z] = meshgrid(x,y,z);
24   U = 0.3+X; V = 0.4-Y; W = 0.5-Z;
25   % quiver3(X,Y,Z,U,V,W)
26   sx = 0;
27   sy = 0:0.1:1;
28   sz = 0:0.1:1;
29   [Sx, Sy, Sz] = meshgrid(sx,sy,sz);
30   SL = stream3(X,Y,Z, U,V,W, Sx,Sy,Sz);
31   streamline(SL)
32   view(3), axis vis3d, box on
33   xlabel('x'), ylabel('y'), zlabel('z')
34   streamparticles(SL, ...
35       'Animate', 5, ...
36       'FrameRate', 30, ...
37       'ParticleAlignment', 'on')
```

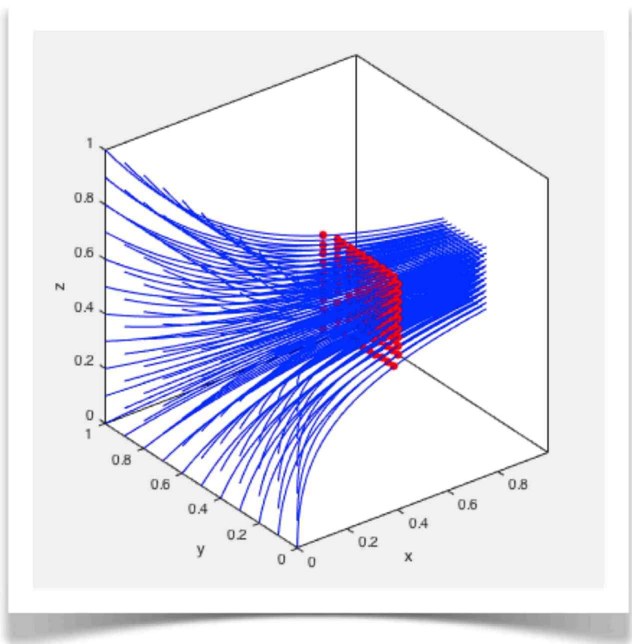


Table 6.2 Stream Particles Functions

Functions	Description
`streamparticles(SL,name,value)`	Plot stream particles
Details and More: Help>MATLAB>Graphics>2-D and 3-D Plots>Surfaces, Volumes, and Polygons> Volume Visualization>Vector Volume Data	

6.3 Movies: Animation of an Engine

Example06_03.m: Animation of an Engine

[1] This program creates a movie that animates the motion of an engine cylinder as shown in [3-7] (next page) and saves the movie as an video file, in default format (AVI). Observe the outcome of each command.

```
clear
...
for k = 1:n
    ...
    Frames(k) = getframe;
    ...
end
movie(Frames, 5, 30)
videoObj = VideoWriter('Engine');
open(videoObj);
writeVideo(videoObj, Frames);
close(videoObj);
```

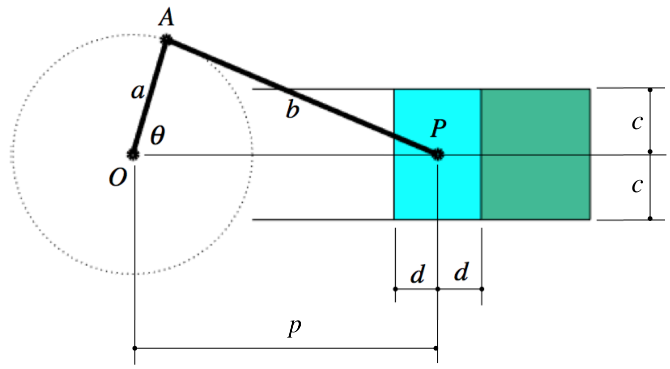


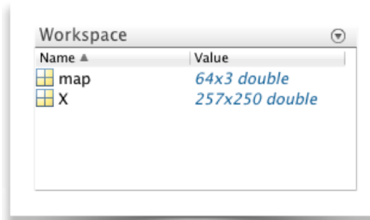
Table 6.3 Animated Line Plots Functions	
Functions	Description
`movie(Frames,n,fps)`	Play movie frames
`frame = getframe`	Capture axes or figure as movie frame
`videoObj = VideoWriter(filename)`	Create object to write video files
`writeVideo(videoObj,Frames)`	Write video data to file
Details and More: Help>MATLAB>Graphics>2-D and 3-D Plots>Animation	

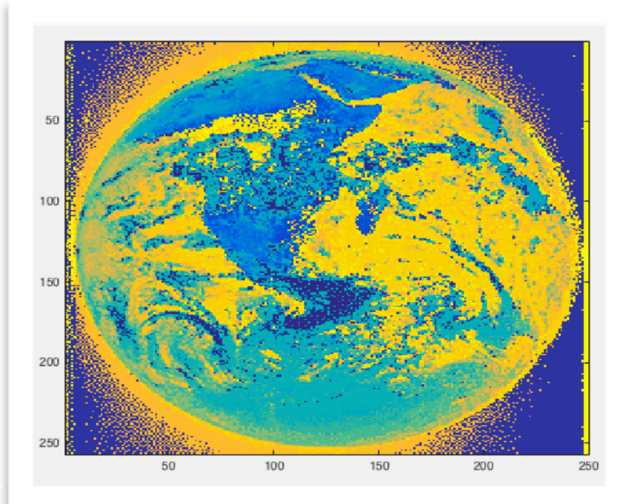
6.4 Indexed Images

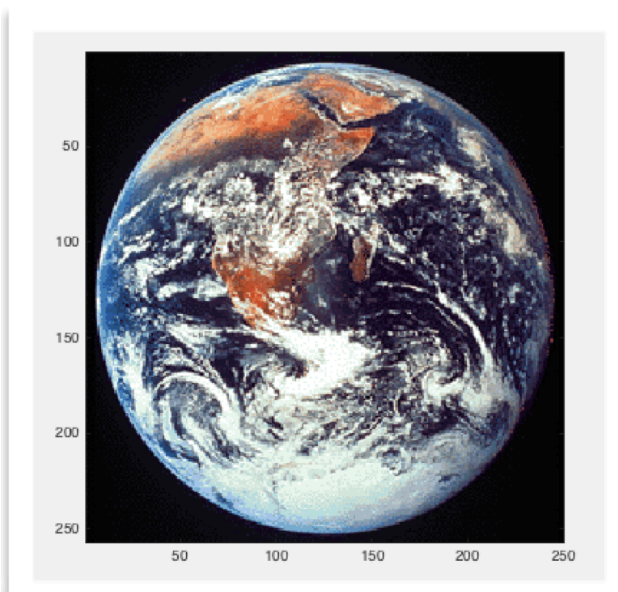
Example06_04a.m: Indexed Images

[1] MATLAB supports two types of images: **indexed** and **true-color**. The following commands demonstrate the display of an indexed image. We'll discuss **true-color** images in the next section.

```
clear
load earth
image(X)
colormap(map)
axis image
```



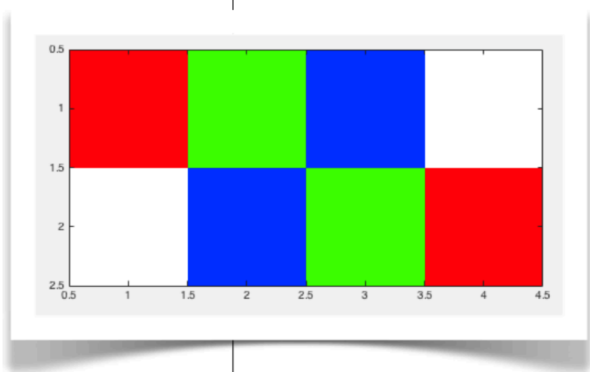




Example06_04b.m: Colormaps and Images

[7] The following script demonstrates the creation of a colormap of 4 colors, the creation of a 2-by-4 image, and some operations of the image. Observe the outcome of each command.

```
clear
MyMap = [1, 0, 0;  % 1. Red
         0, 1, 0;  % 2. Green
         0, 0, 1;  % 3. Blue
         1, 1, 1]; % 4. White
MyImage = [1, 2, 3, 4;
           4, 3, 2, 1];
image(MyImage)
colormap(MyMap)
axis image
delete(gcf)
save('Datafile06_04b','MyImage','MyMap')
clear
load('Datafile06_04b')
image(MyImage)
colormap(MyMap)
axis image
```

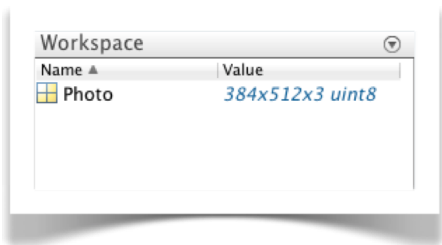


6.5 True-Color Images

Example06_05a.m: True Color Images

[1] We saw the following commands in Section 1.13. It displays a true-color image [2]. Now we look into details of a true-color image.

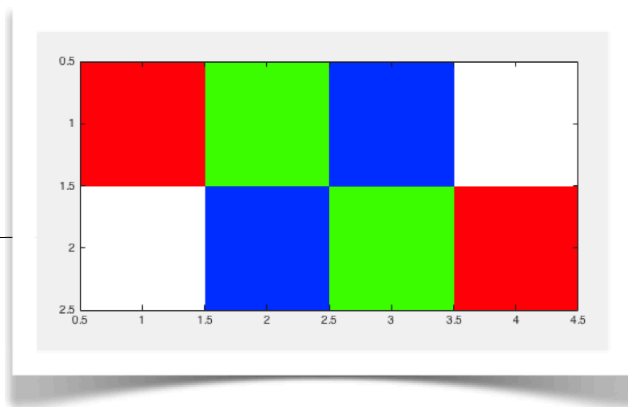
```
clear
Photo = imread('peppers.png');
image(photo)
axis image
```

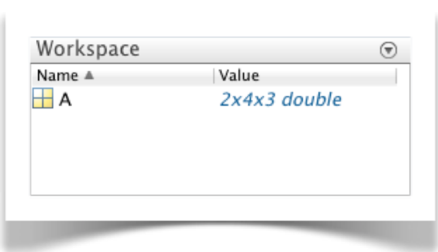


Example06_05b.m: RGB Triplets

[5] The following script demonstrates the creation of a 2-by-4 true-color image and some operations of the image.

```
clear
A = zeros(2,4,3);
A(1,1,1:3) = [1 0 0];  % Red
A(1,2,1:3) = [0 1 0];  % Green
A(1,3,1:3) = [0 0 1];  % Blue
A(1,4,1:3) = [1 1 1];  % White
A(2,1,1:3) = [1 1 1];  % White
A(2,2,1:3) = [0 0 1];  % Blue
A(2,3,1:3) = [0 1 0];  % Green
A(2,4,1:3) = [1 0 0];  % Red
image(A), axis image
imwrite(A, 'Datafile06_05b.png')
delete(gcf), clear
B = imread('Datafile06_05b.png');
image(B), axis image
```





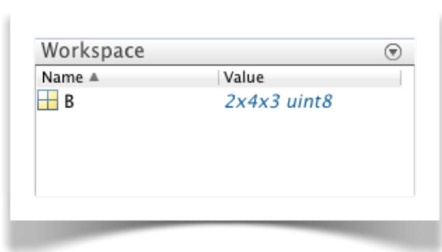


Table 6.5a Supported Image File Formats	
Format	Description
BMP	Microsoft Windows Bitmap
GIF	Graphics Interchange Files
HDF	Hierarchical Data Format
JPEG	Joint Photographic Experts Group
PCX	Paintbrush
PNG	Portable Network Graphics
TIFF	Tagged Image File Format
XWD	X Window Dump
Details and More: Help>MATLAB>Graphics>Images> Working with Images in MATLAB Graphics.	

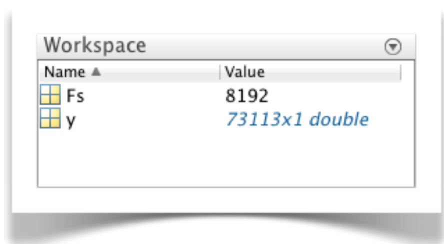
Table 6.5b Image Functions	
Functions	Description
`colormap map`	Set current colormap
`image(A)`	Display image (indexed or true-color)
`[A,map] = imread(filename)`	Read image (indexed or true-color) from file
`imwrite(A,filename)`	Write true-color image to file
`imwrite(A,map,filename)`	Write indexed image to file
`info = imfinfo(filename)`	Get image information
`B = ind2rgb(A,map)`	Convert indexed image to true-color image
Details and More: Help>MATLAB>Graphics>Images	

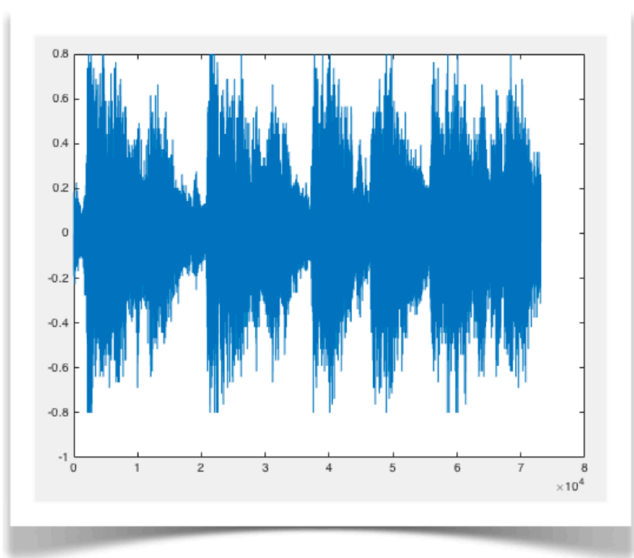
6.6 Audios

Example06_06a.m: Audios

[1] Following commands demonstrate the play and some other operations of an audio data. Observe the outcome of each command.

```
clear
load handel
sound(y, Fs)
plot(y)
audiowrite('handel.wav', y, Fs)
delete(gcf), clear
[y, Fs] = audioread('handel.wav');
plot(y)
sound(y, Fs)
```

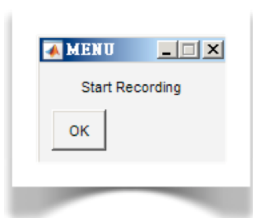


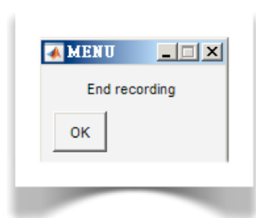


Example06_06b.m: Audio Recording

[5] Assuming your computer has an audio input device, this script records your voice, plays back the voice, saves it in a file, reads it from the file, plays back again, and finally plots the waves of the voice. Observe the outcome of each command.

```
clear
recObj = audiorecorder;
menu('Start Recording', 'OK');
record(recObj)
menu('End recording', 'OK');
stop(recObj)
play(recObj);
y = getaudiodata(recObj);
Fs = recObj.SampleRate;
audiowrite('myvoice.wav', y, Fs)
clear
[y, Fs] = audioread('myvoice.wav');
sound(y, Fs)
plot(y)
```





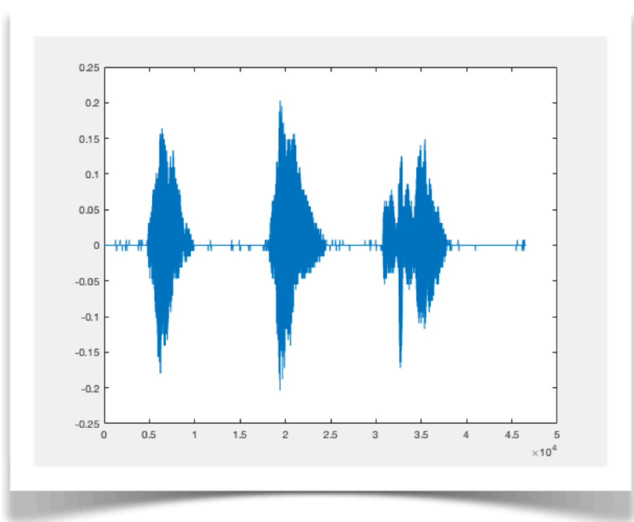


Table 6.6a Supported Audio File Formats

File Extension	Format
`.wav`	WAVE
`.ogg`	OGG
`.flac`	FLAC
`.mp4`	MPEG-4
`.m4a`	AAC
Details and More: `>> doc audiowrite`	

Table 6.6b Audio Functions

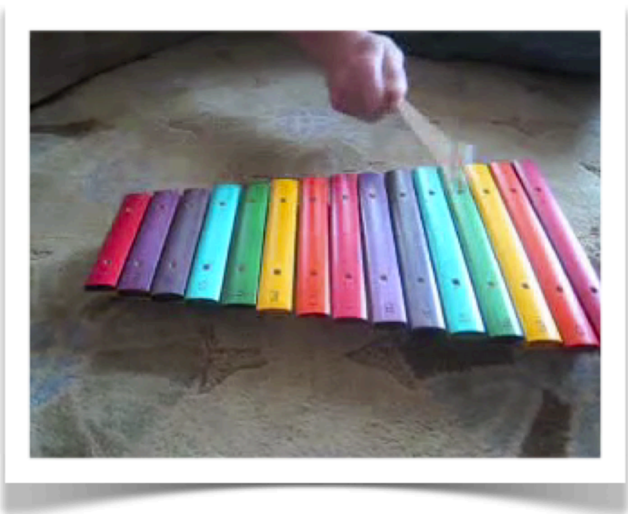
Functions	Description
`[y,Fs] = audioread(filename)`	Read audio file
`audiowrite(filename,y,Fs)`	Write audio file
`recObj = audiorecorder`	Create **audiorecorder** object
`y = getaudiodata(recObj)`	Retrieve recorded data
`record(recObj)`	Record to **audiorecorder** object
`play(recObj)`	Play **audiorecorder** object
`stop(recObj)`	Stop recording
`pause(recObj)`	Pause recording
`resume(recObj)`	Resume recording from paused position
`sound(y,Fs)`	Play sound
Details and More: *Help>MATLAB>Daya Import and Analysis>Data Import and Export>Standard File Formats>Audio and Video*	

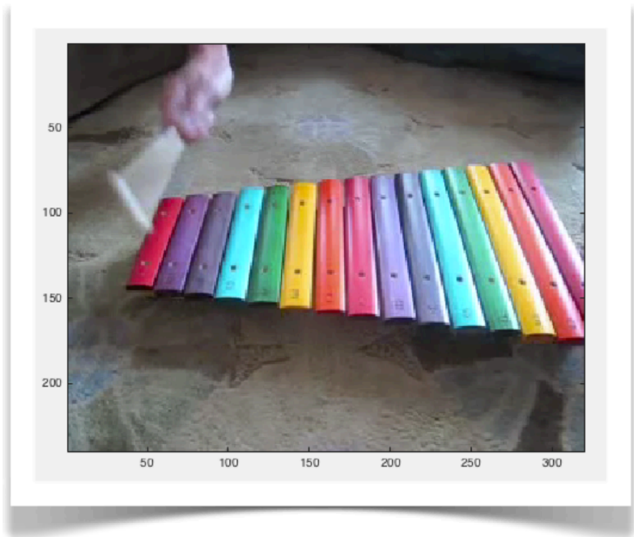
6.7 Videos

Example06_07.m: Videos

[1] The following commands demonstrate the reading of a video file in **mp4** format and the playing of the video using the function `movie` (Section 6.3). A snapshot of the video is shown in [3], next page. (*This example is adapted from the example in Help>MATLAB>Data Import and Analysis>Data Import and Export>Standard File Formats>Audio and Video>Reading and Writing Files>Read Video Files*)

```
clear
vidObj = VideoReader('xylophone.mp4');
height = vidObj.Height;
width  = vidObj.Width;
rate   = vidObj.FrameRate;
Frames.cdata = zeros(height, width, 3, 'uint8');
Frames.colormap = [];
k = 1;
while hasFrame(vidObj)
    Frames(k).cdata = readFrame(vidObj);
    k = k+1;
end
set(gcf, 'Position', [150, 150, width, height])
set(gca, 'Units', 'pixels')
set(gca, 'Position', [0, 0, width, height])
movie(Frames, 1, rate)
```





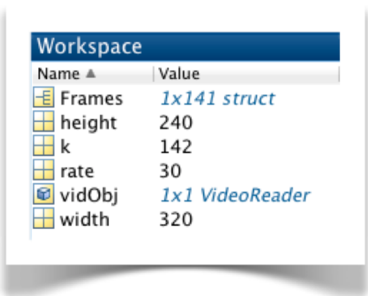


Table 6.7 Video Functions	
Functions	Description
`vidObj = VideoReader(filename)`	Create **VideoReader** object
`vidObj = VideoWriter(filename)`	Create **VideoWriter** objcct
`open(vidObj)`	Open file for writing video data
`close(vidObj)`	Close file after writing video data
`frameData = readFrame(vidObj)`	Read video frame from video file
`writeVideo(vidObj,Frames)`	Write frames to video file
`tf = hasFrame(vidObj)`	Determine if end-of-file reached
`movie(Frames,n,fps)`	Play movie frames
Details and More: *Help>MATLAB>Daya Import and Analysis>Data Import and Export>Standard File Formats>Audio and Video*	

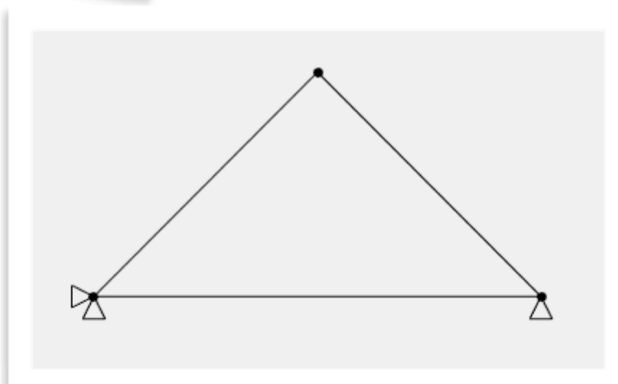
6.8 Example: Statically Determinate Trusses (Version 4.0)

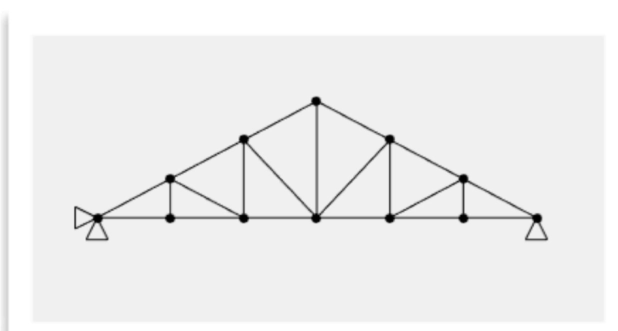
Example06_08.m: Truss 4.0

```
clear
Nodes= struct; Members = struct;
disp(' 1. Input nodal coordinates')
...
disp('10. Quit')
disp('11. Plot truss')
while 1
    task = input('Enter the task number: ');
    switch task
        case 1
            Nodes = inputNodes(Nodes);
        ...
        case 10
            break
        case 11
            plotTruss(Nodes, Members)
    end
end
```

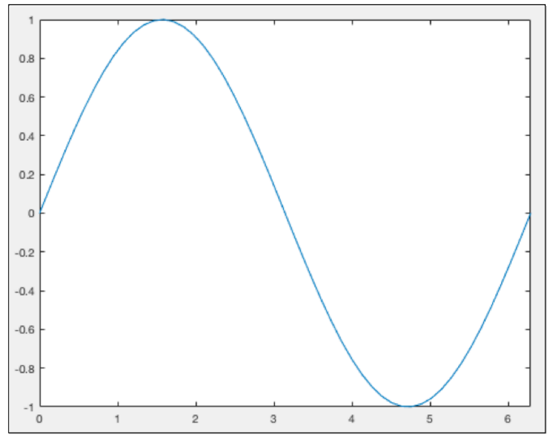
```
function plotTruss(Nodes, Members)
if (size(fieldnames(Nodes),1)<6 || size(fieldnames(Members),1)<2)
    disp('Truss data not complete'); return
end
n = length(Nodes); m = length(Members);
minX = Nodes(1).x; maxX = Nodes(1).x;
minY = Nodes(1).y; maxY = Nodes(1).y;
for k = 2:n
    if (Nodes(k).x < minX) minX = Nodes(k).x; end
    if (Nodes(k).x > maxX) maxX = Nodes(k).x; end
    if (Nodes(k).y < minY) minY = Nodes(k).y; end
    if (Nodes(k).y > maxY) maxY = Nodes(k).y; end
end
rangeX = maxX-minX; rangeY = maxY-minY;
axis([minX-rangeX/5, maxX+rangeX/5, minY-rangeY/5, maxY+rangeY/5])
ha = gca; delete(ha.Children)
axis equal off
hold on
for k = 1:m
    n1 = Members(k).node1; n2 = Members(k).node2;
    x = [Nodes(n1).x, Nodes(n2).x];
    y = [Nodes(n1).y, Nodes(n2).y];
    plot(x,y,'k-o', 'MarkerFaceColor', 'k')
end
for k = 1:n
    if Nodes(k).supportx
        x = [Nodes(k).x, Nodes(k).x-rangeX/20, Nodes(k).x-rangeX/20, Nodes(k).x];
        y = [Nodes(k).y, Nodes(k).y+rangeX/40, Nodes(k).y-rangeX/40, Nodes(k).y];
        plot(x,y,'k-')
    end
    if Nodes(k).supporty
        x = [Nodes(k).x, Nodes(k).x-rangeX/40, Nodes(k).x+rangeX/40, Nodes(k).x];
        y = [Nodes(k).y, Nodes(k).y-rangeX/20, Nodes(k).y-rangeX/20, Nodes(k).y];
        plot(x,y,'k-')
    end
end
end
```

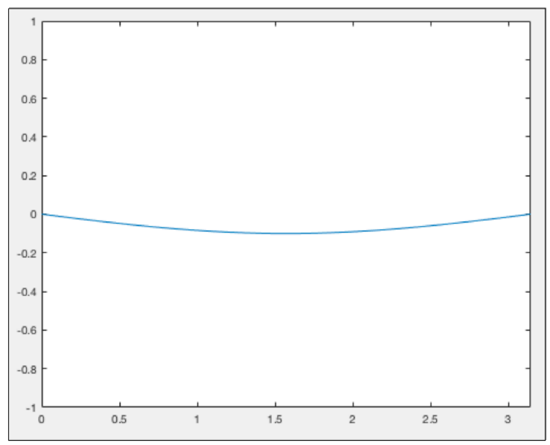
```
>> Example06_08
 1. Input nodal coordinates
 2. Input connecting nodes of members
 3. Input three supports
 4. Input loads
 5. Print truss
 6. Solve truss
 7. Print results
 8. Save data
 9. Load data
10. Quit
11. Plot truss
Enter the task number: 9
Enter file name (default Datafile): Datafile04_04a
Enter the task number: 11
Enter the task number: 9
Enter file name (default Datafile): Datafile04_04b
Enter the task number: 11
Enter the task number: 10
>>
```

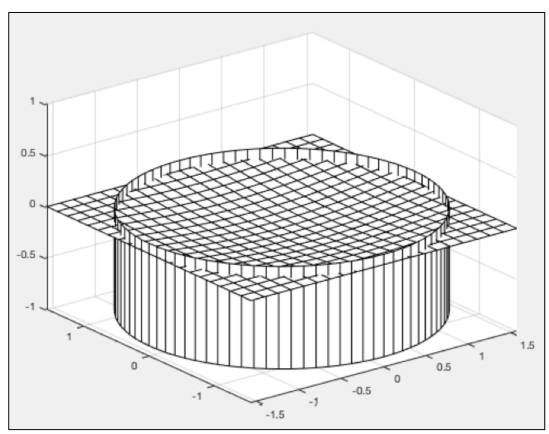


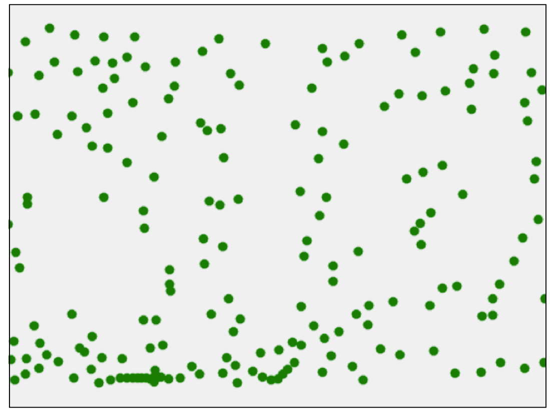


6.9 Additional Exercise Problems









Chapter 7

Data Import and Export

A program can be viewed as a data processing system. Data can be exported to or imported from a device (e.g., keyboard) or a disk file. We've seen many examples in which data are input from the keyboard and output to the computer screen. We've also seen many examples in which data are read from and written to files. In this chapter, we'll discuss details and more about data import and export.

7.1 Screen Text I/O

Example07_01a.m: `input` and `disp`

[1] The function `input` reads data from your computer screen and the function `disp` writes data to the computer screen. Run this script and enter data as shown in [2].

```
clear
while 1
    a = input('Enter anything: ');
    if strcmp(a, 'stop') break, end
    disp(class(a))
    disp(a)
end
```

```
>> Example07_01a
Enter anything: 56
double
    56
Enter anything: 3+11
double
    14
Enter anything: 5*sin(pi/5)^2
double
    1.7275
Enter anything: int16(56)
int16
     56
Enter anything: 'b'
char
b
Enter anything: 'bcdef'
char
bcdef
Enter anything: b
Error using input
Undefined function or variable 'b'.
Error in Example07_01a (line 3)
    a = input('Enter anything: ');
Enter anything: bcdef
Error using input
Undefined function or variable 'bcdef'.
Error in Example07_01a (line 3)
    a = input('Enter anything: ');
Enter anything: 'stop'
>>
```

Function `disp`

[4] A syntax for the function `disp` is

`disp(`*`var`*`)`

It displays the value of *`var`*, which can be any type (class) of data. Actually, each class of data has its definition of `disp`. We've demonstrated this in Section 4.9 (also see 4.9[10], page 203).

These are some more examples:

```
39   >> a = [3, 6, 7, 2];
40   >> disp(a)
41        3     6     7     2
42   >> c = 6; d = 5;
43   >> disp(c>d)
44        1
```

In line 43, the result of `c>d` is of class `logical`, hence it displays a logical value 1 (line 44). →

```
>> clear
>> a = pi*10000; b = -314;
>> fprintf('%f %f\n', a, b)
31415.926536 -314.000000
>> fprintf('%.4f %.4f\n', a, b)
31415.9265 -314.0000
>> fprintf('%12f %12f\n', a, b)
31415.926536  -314.000000
>> fprintf('%12.4f %12.4f\n', a, b)
  31415.9265    -314.0000
>> fprintf('%8.4f %8.4f\n', a, b)
31415.9265 -314.0000
>> fprintf('%-12.4f %-12.4f\n', a, b)
31415.9265   -314.0000
>> fprintf('%+12.4f %+12.4f\n', a, b)
 +31415.9265    -314.0000
>> fprintf('%e %e\n', a, b)
3.141593e+04 -3.140000e+02
>> fprintf('%.4e %.4e\n', a, b)
3.1416e+04 -3.1400e+02
>> fprintf('%12.4e %12.4e\n', a, b)
  3.1416e+04  -3.1400e+02
>> fprintf('%g %g\n', a, b)
31415.9 -314
>> fprintf('%.4g %.4g\n', a, b)
3.142e+04 -314
>> fprintf('%12.4g %12.4g\n', a, b)
   3.142e+04         -314
>> fprintf('%d %d\n', b, b)
-314 -314
>> fprintf('%5d %5d\n', b, b)
 -314  -314
>> fprintf('%c %c\n', 'A', 'B')
A B
>> fprintf('%c %c\n', 65, 66)
A B
>> fprintf('%5c %5c\n', 'A', 'B')
    A     B
>> fprintf('%s %s\n','A','string')
A string
>> fprintf('%8s %15s\n','A','string')
       A          string
>> fprintf('%3s %3s\n','A','string')
  A string
```

Table 7.1a Examples of Format Specifications

Format	Description
`%8.4f`	Fixed-point notation
`%8.4e`	Exponential notation
`%8.4g`	The more compact of %f or %e
`%8d`	Signed integer
`%8u`	Unsigned integer
`%8c`	Character
`%8s`	String

Details and More: Help>MATLAB>Data Import and Analysis>Data Import and Export>Lower-Level File I/O>fprintf>Input Arguments>formatSpec

Table 7.1b Screen Text I/O

Functions	Description
`x = input(prompt, 's')`	Request user input
`disp(x)`	Display value of variable
`fprintf(format,a,b,...)`	Write text data to screen or file
`s = sprintf(format,a,b,...)`	Format text data into string

7.2 Low-Level Text File I/O

Example07_02.m: Text-File Explorer

[1] This program, a "text-file explorer," is designed to demonstrate many text-file I/O functions (see Table 7.2, page 285). We'll use the file Example07_01a.m (page 278, which is a text file of 124 characters; see a copy in [3], next page) as the target to test this program. Before looking into each statement, test run this program as shown in [4], page 283.

```
clear
fileName = input('Enter the file name: ', 's');
fileID = fopen(fileName);
disp('0. stop')
disp('1. Read the file once for all')
disp('2. Read the file one line at a time')
disp('3. Read a line')
disp('4. Read a character')
disp('5. Rewind to the beginning')
disp('6. Move forward a character')
disp('7. Move backward a character')
while 1
    task = input('Enter task number: ');
    switch task
        case 0
            fclose(fileID);
            break
        case 1
            text = fileread(fileName);
            disp(text)
            characters = length(text);
            disp([num2str(characters), ' characters read'])
        case 2
            frewind(fileID), lines = 0; characters = 0;
            while ~feof(fileID)
                text = fgetl(fileID);
                disp(text)
                lines = lines+1;
                characters = characters + length(text);
            end
            disp([num2str(lines), ' lines read'])
            disp([num2str(characters), ' characters read'])
        case 3
            if feof(fileID)
                disp('End of file!')
            else
                text = fgetl(fileID);
                disp(text);
            end
```

(Continued at [2], next page) →

[2] Example07_02.m (Continued).

```
        case 4
            if feof(fileID)
                disp('End of file!')
            else
                text = fscanf(fileID, '%c', 1);
                disp(text);
            end
        case 5
            frewind(fileID)
        case 6
            if feof(fileID)
                disp('End of file!')
            else
                fseek(fileID, 1, 'cof');
            end
        case 7
            if ftell(fileID) == 0
                disp('Beginning of file!')
            else
                fseek(fileID, -1, 'cof');
            end
    end
    position = ftell(fileID);
    disp(['File pointer at ', num2str(position)])
end
```

```
clear
while 1
    a = input('Enter anything: ');
    if strcmp(a, 'stop') break, end
    disp(class(a))
    disp(a)
end
```

```
>> Example07_02
Enter the file name: Example07_01a.m
0. stop
1. Read the file once for all
2. Read the file one line at a time
3. Read a line
4. Read a character
5. Rewind to the beginning
6. Move forward a character
7. Move backward a character
Enter task number: 1
clear
while 1
    a = input('Enter anything: ');
    if strcmp(a, 'stop') break, end
    disp(class(a))
    disp(a)
end
119 characters read
File pointer at 0
Enter task number: 2
clear
while 1
    a = input('Enter anything: ');
    if strcmp(a, 'stop') break, end
    disp(class(a))
    disp(a)
end
7 lines read
113 characters read
File pointer at 119
Enter task number: 5
File pointer at 0
Enter task number: 3
clear
File pointer at 6
Enter task number: 3
while 1
File pointer at 14
Enter task number: 5
File pointer at 0
Enter task number: 4
c
File pointer at 1
Enter task number: 6
File pointer at 2
Enter task number: 4
e
File pointer at 3
Enter task number: 4
a
File pointer at 4
Enter task number: 7
File pointer at 3
Enter task number: 4
a
File pointer at 4
Enter task number: 3
r
File pointer at 6
Enter task number: 0
>>
```

Table 7.2 Low-Level Text File I/O

Functions	Description
`fileID = fopen(filename,permission)`	Open file
`fclose(fileID)`	Close one or all open files
`tf = feof(fileID)`	Test for end-of-file
`[message,errnum] = ferror(fileID)`	Information about file I/O errors
`s = fgetl(fileID)`	Read line from file, removing newline character
`s = fgets(fileID)`	Read line from file, keeping newline character
`s = fileread(filename)`	Read contents of file into string
`fprintf(fileID,format,a,b,...)`	Write data to text file
`frewind(fileID)`	Move file position indicator to beginning of open file
`a = fscanf(fileID,format)`	Read formatted data from text file
`fseek(fileID,offset,origin)`	Move to specified position in file
`position = ftell(fileID)`	Position in open file
`type filename`	Display contents of file

Details and More: Help>MATLAB>Data Import and Analysis>Data Import and Export>Low-Level File I/O

7.3 Low-Level Binary File I/O

Byte	Bits	Bit Pattern 7654 3210	uint8 Value
1	0-7	0000 0000	0
2	8-15	0000 0000	0
3	16-23	0000 0000	0
4	24-31	0000 0000	0
5	32-39	0000 0000	0
6	40-47	0000 0000	0
7	48-55	0011 1100	60
8	56-63	0100 0000	64

Example07_03.m: Bit Pattern

[2] This script confirms the concepts presented in [1]. →

```
clear
a = 28;
fileID = fopen('tmp.dat','w+');
fwrite(fileID, a, 'double');
frewind(fileID)
b = fread(fileID, 8, 'uint8');
disp(b')
fclose(fileID);
delete('tmp.dat')
```

Table 7.3 Low-Level Binary File I/O	
Functions	Description
`fileID = fopen(filename,permission)`	Open file
`fclose(fileID)`	Close one or all open files
`a = fread(fileID,size)`	Read data from binary file
`fwrite(fileID,a)`	Write data to binary file
`tf = feof(fileID)`	Test for end-of-file
`[message,errnum] = ferror(fileID)`	Information about file I/O errors
`frewind(fileID)`	Move file position indicator to beginning of open file
`fseek(fileID,offset,origin)`	Move to specified position in file
`position = ftell(fileID)`	Position in open file
Details and More: Help>MATLAB>Data Import and Analysis>Data Import and Export>Low-Level File I/O	

7.4 MAT-Files

Example07_04.m: Out-of-Core Matrices

[2] This script demonstrate the ideas in [1].

```
clear
M = matfile('tmp');
n = 10;
for i = 1:n
    for j = 1:n
        M.A(i,j) = i+(j-1)*n;
    end
end
clear
load('tmp')
delete('tmp.mat')
disp(A)
```

```
     1    11    21    31    41    51    61    71    81    91
     2    12    22    32    42    52    62    72    82    92
     3    13    23    33    43    53    63    73    83    93
     4    14    24    34    44    54    64    74    84    94
     5    15    25    35    45    55    65    75    85    95
     6    16    26    36    46    56    66    76    86    96
     7    17    27    37    47    57    67    77    87    97
     8    18    28    38    48    58    68    78    88    98
     9    19    29    39    49    59    69    79    89    99
    10    20    30    40    50    60    70    80    90   100
```

Table 7.4 MAT-Files

Functions	Description
`load(filename)`	Load variables from file into Workspace
`save(filename,v1,v2,...)`	Save variables to file
`save filename`	Save all variables to file
`M = matfile(filename)`	Access variables in MAT-files, without loading into memory.

Details and More:
Help>MATLAB>Data Import and Analysis>Data Import and Export>Workspace Variables and MAT-Files

7.5 ASCII-Delimited Files

Example07_05.m

[1] An ASCII-delimited file is a text file containing data separated by delimiters such as space, comma, tab, semicolon, newline, etc. This script demonstrates the manipulation of ASCII-delimited files.

```
clear
type count.dat
A = dlmread('count.dat');
dlmwrite('tmp.dat', A)
type tmp.dat
clear
B = csvread('tmp.dat');
delete tmp.dat
csvwrite('tmp1.dat', B)
type tmp1.dat
delete tmp1.dat
```

```
>> type count.dat

    11    11     9
     7    13    11
    14    17    20
    11    13     9
    43    51    69
    38    46    76
    61   132   186
    75   135   180
    38    88   115
    28    36    55
    12    12    14
    18    27    30
    18    19    29
    17    15    18
    19    36    48
    32    47    10
    42    65    92
    57    66   151
    44    55    90
   114   145   257
    35    58    68
    11    12    15
    13     9    15
    10     9     7
```

```
>> type tmp.dat

11,11,9
7,13,11
14,17,20
11,13,9
43,51,69
38,46,76
61,132,186
75,135,180
38,88,115
28,36,55
12,12,14
18,27,30
18,19,29
17,15,18
19,36,48
32,47,10
42,65,92
57,66,151
44,55,90
114,145,257
35,58,68
11,12,15
13,9,15
10,9,7
```

Table 7.5 ASCII-Delimited Files	
Functions	Description
`M = dlmread(filename)`	Read ASCII-delimited file into matrix
`dlmwrite(filename,M)`	Write matrix to ASCII-delimited file
`M = csvread(filename)`	Read comma-separated value file (CSV)
`csvwrite(filename,M)`	Write comma-separated value file (CSV)
Details and More: *Help>MATLAB>Data Import and Analysis>Data Import and Export>Standard File Formats>Text Files*	

7.6 Excel Spreadsheet Files

Example07_06.m: Excel Files

[1] MATLAB provides functions that write to (`xlswrite`) and read from (`xlsread`) **Microsoft Excel** spreadsheet files (`.xls` or `.xlsx` files). To use these functions, you need to have Microsoft Excel installed in your computer. This script demonstrates the use of these functions.

```
clear
A = reshape(1:15, 5, 3);
xlswrite('tmp', A, 'Sheet1', 'A2')
title = {'First', 'Second', 'Third'};
xlswrite('tmp', title, 'Sheet1', 'A1:C1')
clear
[num, txt] = xlsread('tmp', 'Sheet1')
delete('tmp.xls')
```

```
num =
     1     6    11
     2     7    12
     3     8    13
     4     9    14
     5    10    15
txt =
  1×3 cell array
    'First'    'Second'    'Third'
```

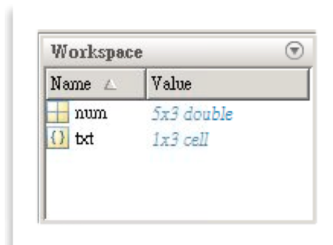
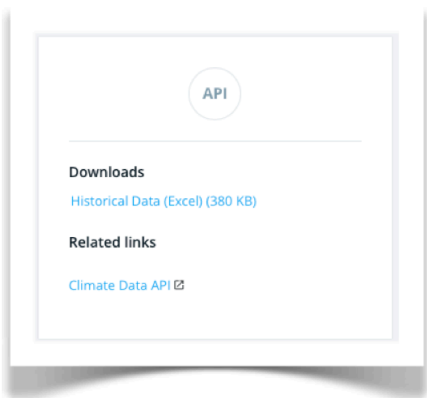
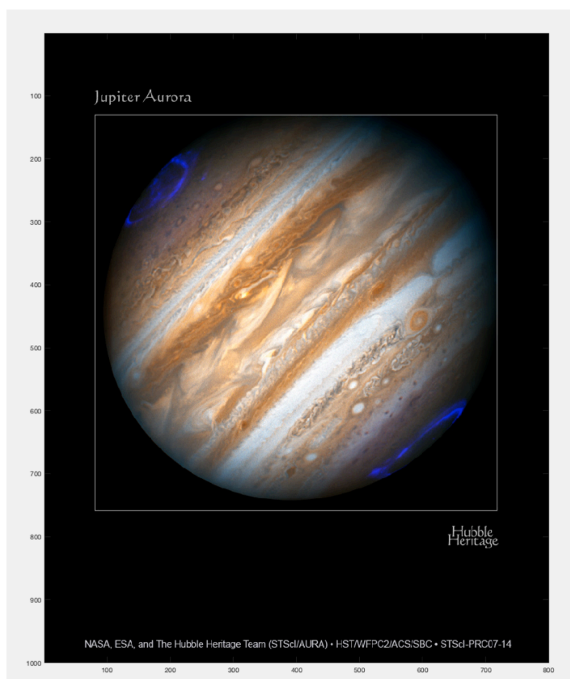
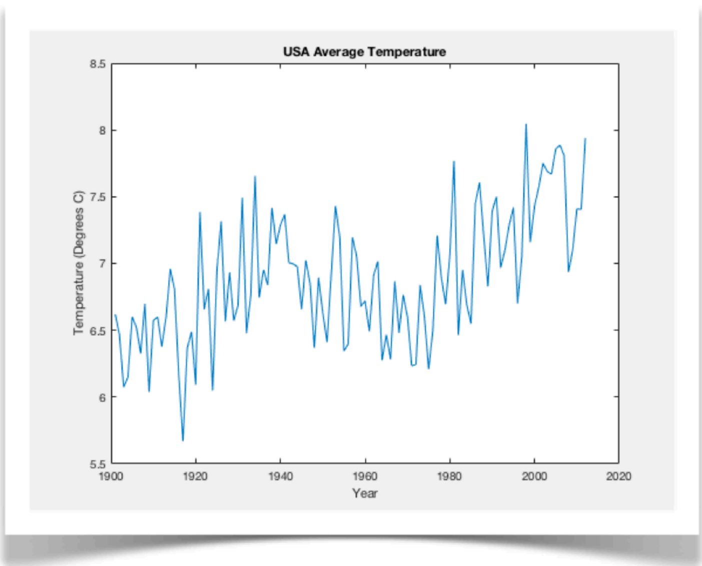


Table 7.6 Excel Spreadsheet Files	
Functions	Description
`[num,txt] = xlsread(filename,sheet,range)`	Read Microsoft Excel spreadsheet file
`xlswrite(filename,M,sheet,range)`	Write Microsoft Excel spreadsheet file
Details and More: Help>MATLAB>Data Import and Analysis>Data Import and Export>Standard File Formats>Spreadsheets	

7.7 Additional Exercise Problems






USA Average Temperature
Temperature (Degrees C)
Year
8.5
8
7.5
7
6.5
6
5.5
1900
1920
1940
1960
1980
2000
2020

Chapter 8

Graphical User Interfaces

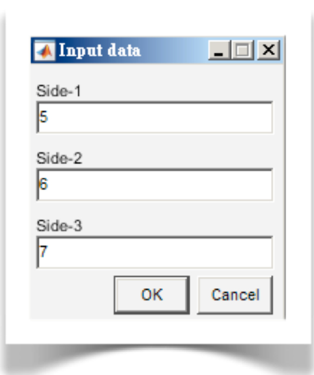
User interface design is a crucial part of programming activities. It is not uncommon that a programmer spends a majority of programming time designing the graphical user interface of a program. It is also not uncommon that a software package fails in the market just because it has a poor user interface design, even if it has excellent functionalities.

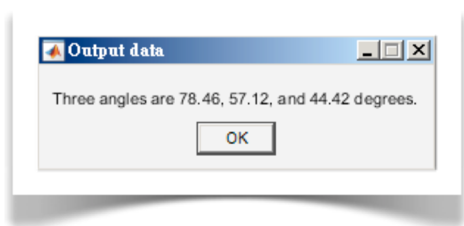
8.1 Predefined Dialog Boxes

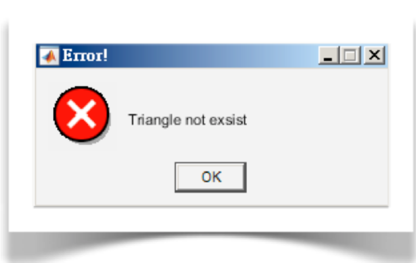
Example08_01a.m: Triangle

[1] MATLAB provides many predefined dialog boxes (Table 8.1, page 299). The following script demonstrates some of these predefined dialog boxes. This script requests the user for three sides of a triangle as shown in [2], calculates the three angles according to the Law of Cosine (2.11[6], page 106), and displays the result as shown in [3]. If the three sides do not satisfy the **triangle inequality** (explained in [4], next page), an **error dialog box** as shown in [5] (next page) is displayed.

```
clear
answer = inputdlg({'Side-1', 'Side-2', 'Side-3'}, ...
    'Input data', 1, {'5', '6', '7'});
s = str2double(answer);
s = sort(s); a = s(1); b = s(2); c = s(3);
if (a+b) <= c
    errordlg('Triangle not exsist', 'Error!', 'modal')
else
    alpha = acosd((b^2+c^2-a^2)/(2*b*c));
    beta  = acosd((c^2+a^2-b^2)/(2*c*a));
    gamma = acosd((a^2+b^2-c^2)/(2*a*b));
    message = sprintf('Three angles are %.2f, %.2f, and %.2f degrees.', ...
        alpha, beta, gamma);
    msgbox(message, 'Output data', 'modal')
end
```






Error!
Triangle not exsist
OK

Example08_01b.m: Voice Recorder

[6] This script, an enhanced version of Example06_06b.m (page 268), uses additional predefined dialog boxes. It displays a menu of 5 tasks for you to choose [7-8]: start the recording, end the recording, play the voice, save the voice as a sound file, and load a sound file.

```
clear
y = [];
while 1
    choice = menu('Voice Recorder', ...
        'Start', 'End', 'Play', 'Save', 'Load');
    switch choice
        case 0  % The user clicks the close button
            break
        case 1  % Start recording
            recObj = audiorecorder;
            record(recObj)
        case 2  % End recording
            stop(recObj)
            y = getaudiodata(recObj);
            Fs = recObj.SampleRate;
        case 3  % Play
            if isempty(y)
                errordlg('Empty!','Error!','modal')
            else
                sound(y, Fs);
            end
        case 4  % Save
            if isempty(y)
                errordlg('Empty!','Error!','modal')
            else
                [file, path] = uiputfile('*.wav');
                if file
                    audiowrite([path, file], y, Fs)
                end
            end
        case 5  % Load
            [file, path] = uigetfile('*.wav');
            if file
                [y, Fs] = audioread([path, file]);
            end
    end
end
```

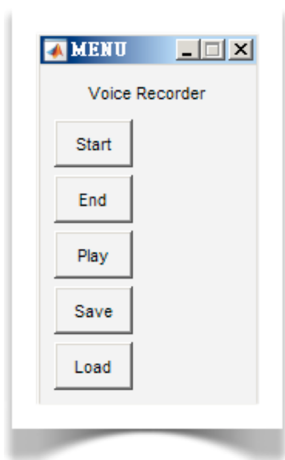


Table 8.1 Predefined Dialog Boxes

Functions	Description
`errordlg(message,title,mode)`	Create error dialog box
`warndlg(message,title,mode)`	Create warning dialog box
`msgbox(message,title,mode)`	Create message dialog box
`waitbar(x,message)`	Open or update wait bar dialog box
`button = questdlg(message,title)`	Create yes-no-cancel dialog box
`answer = inputdlg(prompt,title,n,default)`	Create dialog box that gathers user input
`choice = listdlg(name,value)`	Create list-selection dialog box
`[file,path] = uigetfile(filterSpec)`	Open file-selection dialog box
`[file,path] = uiputfile(filterSpec)`	Open dialog box for saving files
`choice = menu(message,op1,op2,...)`	Create multiple-choice dialog box

Details and More: Help>MATLAB>App Building>GUIDE or Programmatic Workflow>Dialog Boxes

8.2 UI-Controls: Pushbuttons

```
voiceRecorder

function voiceRecorder
y = []; Fs = 0; recObj = [];
figure('Position', [300, 300, 200, 300], ...
    'Name', 'Voice Recorder', ...
    'MenuBar', 'none', ...
    'NumberTitle', 'off');
uicontrol('Style', 'pushbutton', ...
    'String', 'Start', ...
    'Position', [50, 250, 100, 20], ...
    'Callback', @cbStart)
uicontrol('Style', 'pushbutton', ...
    'String', 'End', ...
    'Position', [50, 210, 100, 20], ...
    'Callback', @cbEnd)
uicontrol('Style', 'pushbutton', ...
    'String', 'Play', ...
    'Position', [50, 170, 100, 20], ...
    'Callback', @cbPlay)
uicontrol('Style', 'pushbutton', ...
    'String', 'Save', ...
    'Position', [50, 130, 100, 20], ...
    'Callback', @cbSave)
uicontrol('Style', 'pushbutton', ...
    'String', 'Load', ...
    'Position', [50, 90, 100, 20], ...
    'Callback', @cbLoad)
uicontrol('Style', 'pushbutton', ...
    'String', 'Quit', ...
    'Position', [50, 50, 100, 20], ...
    'Callback', @cbQuit)

    function cbStart(~, ~)
        recObj = audiorecorder;
        record(recObj)
    end

    function cbEnd(~, ~)
        stop(recObj)
        y = getaudiodata(recObj);
        Fs = recObj.SampleRate;
    end

    function cbPlay(~, ~)
        if isempty(y)
            errordlg('Empty voice!', 'modal')
        else
            sound(y, Fs);
        end
    end

```

```
    function cbSave(~, ~)
        if isempty(y)
            errordlg('Empty voice!', 'modal')
        else
            [file, path] = uiputfile('*.wav');
            if file
                audiowrite([path, file], y, Fs)
            end
        end
    end

    function cbLoad(~, ~)
        [file, path] = uigetfile('*.wav');
        if file
            [y, Fs] = audioread([path, file]);
        end
    end

    function cbQuit(~, ~)
        close
    end
end
```

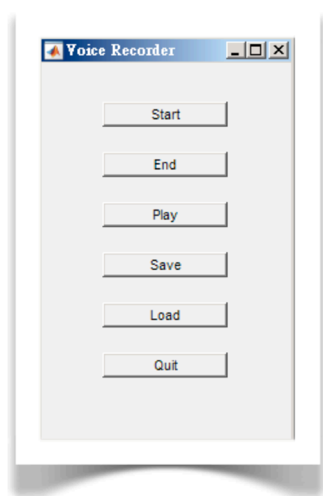


Table 8.2a UI Control Properties

Property	Description
`Style`	Style of UI control. (See Table 8.2b)
`Parent`	Parent of uicontrol.
`Position`	Location and size of UI control, [left, bottom, width, height].
`Units`	Units of measurement. (Default: pixels)
`FontSize`	Font size for text, a positive number.
`String`	Text to display.
`BackgroundColor`	Background color of uicontrol
`HorizontalAlignment`	Alignment of text.
`Callback`	Callback function when user interacts with uicontrol
`ButtonDownFcn`	Button-press callback function
`UIContextMenu`	Uicontrol context menu.
`Value`	Current value of uicontrol.
`Max`	Maximum value of uicontrol.
`Min`	Minimum value of uicontrol.
`SliderStep`	Slider step size.
`Enable`	Operational state of uicontrol

Details and More: Help>MATLAB>App Building>GUIDE or Programmatic Workflow> Components and Layout>Interactive Components>Uicontrol Properties

Table 8.2b Style of UI Control

Style	Description
`pushbutton`	Button that appears to depress until you release the mouse button.
`togglebutton`	The state of a toggle button changes every time you click it.
`checkbox`	The state of a toggle button changes every time you click it.
`radiobutton`	Radio buttons are intended to be mutually exclusive within a group of buttons.
`edit`	Editable text field.
`text`	Static text field.
`slider`	The position of a slider button indicates a value within a range.
`listbox`	List of items from which the user can select one or more items.
`popupmenu`	Menu that expands to display a list of choices.

Details and More: Help>MATLAB>App Building>GUIDE or Programmatic Workflow> Components and Layout>Interactive Components>Uicontrol Properties>Type of Control>Style

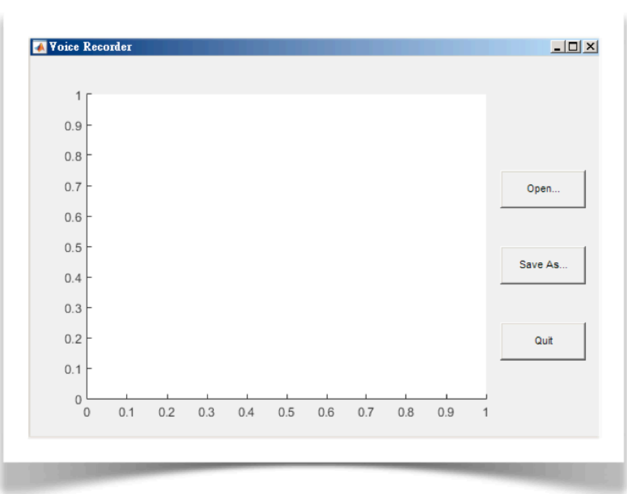
8.3 Example: Image Viewer

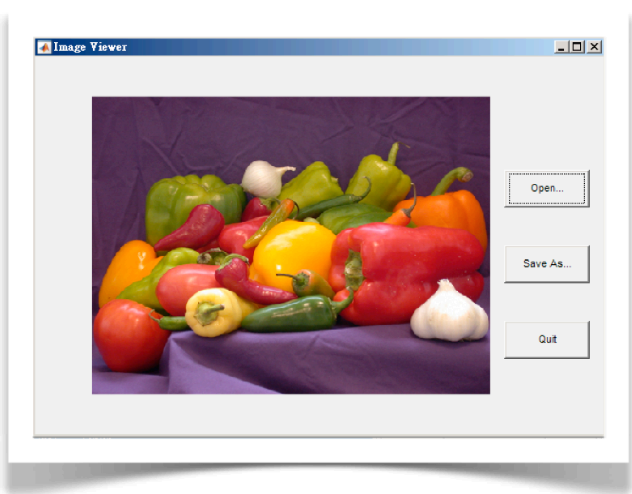
Example08_03.m: Image Viewer

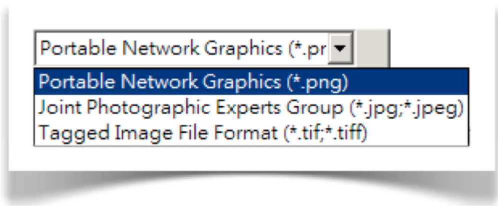
[1] This program opens an image file and displays it on a **Figure** window ([4-6], next page). It also can save the file as another file name. This program demonstrates a more sophisticated use of functions `uigetfile` and `uiputfile` ([5], page 306), which we used in Example08_01b.m, page 298. Locations and sizes of the **Axes** and the **pushbuttons** are specified in normalized units, so the **Figure** window can be resized without twisting the components in the **Figure** window [6]. (Continued at [2], next page.) →

```
imageViewer

function imageViewer
Photo = [];
figure('Position', [30, 30, 600, 400], ...
    'Name', 'Image Viewer', ...
    'MenuBar', 'none', ...
    'NumberTitle', 'off');
axes('Position', [.1 .1 .7 .8]);
uicontrol('Style', 'pushbutton', ...
    'String', 'Open...', ...
    'Callback', @cbOpen, ...
    'Units', 'normalized', ...
    'Position', [.825, .6, .15, .1])
hSaveAs = uicontrol('Style', 'pushbutton', ...
    'String', 'Save As...', ...
    'Callback', @cbSaveAs, ...
    'Enable', 'off', ...
    'Units', 'normalized', ...
    'Position', [.825, .4, .15, .1]);
uicontrol('Style', 'pushbutton', ...
    'String', 'Quit', ...
    'Callback', 'close', ...
    'Units', 'normalized', ...
    'Position', [.825, .2, .15, .1])

    function cbOpen(~, ~)
        [file, path] = uigetfile( ...
        {'*.png', 'Portable Network Graphics (*.png)'; ...
         '*.jpg;*.jpeg', 'Joint Photographic Experts Group (*.jpg;*.jpeg)'; ...
         '*.tif;*.tiff', 'Tagged Image File Format (*.tif;*.tiff)'});
        if file
            Photo = imread([path, file]);
            image(Photo);
            axis off image
            hSaveAs.Enable = 'on';
        end
    end

```

```
    function cbSaveAs(~, ~)
        [file, path] = uiputfile( ...
        {'*.png', 'Portable Network Graphics (*.png)'; ...
         '*.jpg', 'Joint Photographic Experts Group (*.jpg)'; ...
         '*.tif', 'Tagged Image File Format (*.tif)'});
        if file
            imwrite(Photo, [path, file])
        end
    end
end
```






Portable Network Graphics (*.pr
Portable Network Graphics (*.png)
Joint Photographic Experts Group (*.jpg;*.jpeg)
Tagged Image File Format (*.tif;*.tiff)

8.4 UI-Menus: Image Viewer

Example08_04.m: Image Viewer (UI-Menus)

[1] Another way to implement a GUI for the Imager Viewer is using pulldown menus. This program, modified from Example08_03.m (pages 304-305), demonstrates the use of the UI-menus, including nested menus (submenus) asshown in [2], next page. Note that the callback function `cbOpen` (lines 18-29) is the same as that in Example08_03.m. →

```
 1  imageViewer
 2
 3  function imageViewer
 4  Photo = [];
 5  figure('Position', [30, 30, 600, 400], ...
 6      'Name', 'Image Viewer', ...
 7      'ToolBar', 'none', ...
 8      'NumberTitle', 'off');
 9  axes('Position', [.15 .1 .7 .8]);
10  hImage = uimenu('Label', 'Image');
11      uimenu(hImage, 'Label', 'Open...', 'Callback', @cbOpen)
12      hSaveAs = uimenu(hImage, 'Label', 'Save As', 'Enable', 'off');
13          hPNG = uimenu(hSaveAs, 'Label', 'PNG', 'Callback', @cbSaveAs);
14          hJPG = uimenu(hSaveAs, 'Label', 'JPG', 'Callback', @cbSaveAs);
15          hTIF = uimenu(hSaveAs, 'Label', 'TIF', 'Callback', @cbSaveAs);
16      uimenu(hImage, 'Label', 'Quit', 'Callback', 'close')
17
18      function cbOpen(~, ~)
19          [file, path] = uigetfile( ...
20          {'*.png', 'Portable Network Graphics (*.png)'; ...
21           '*.jpg;*.jpeg', 'Joint Photographic Experts Group (*.jpg;*.jpeg)'; ...
22           '*.tif;*.tiff', 'Tagged Image File Format (*.tif;*.tiff)'});
23          if file
24              Photo = imread([path, file]);
25              image(Photo);
26              axis off image
27              hSaveAs.Enable = 'on';
28          end
29      end
30
31      function cbSaveAs(h, ~)
32          if h == hPNG
33              [file,path] = uiputfile({'*.png','Portable Network Graphics (*.png)'});
34          elseif h == hJPG
35              [file,path] = uiputfile({'*.jpg','Joint Photographic Experts Group (*.jpg)'});
36          else
37              [file,path] = uiputfile({'*.tif','Tagged Image File Format (*.tif)'});
38          end
39          if file
40              imwrite(Photo, [path, file])
41          end
42      end
43  end
```

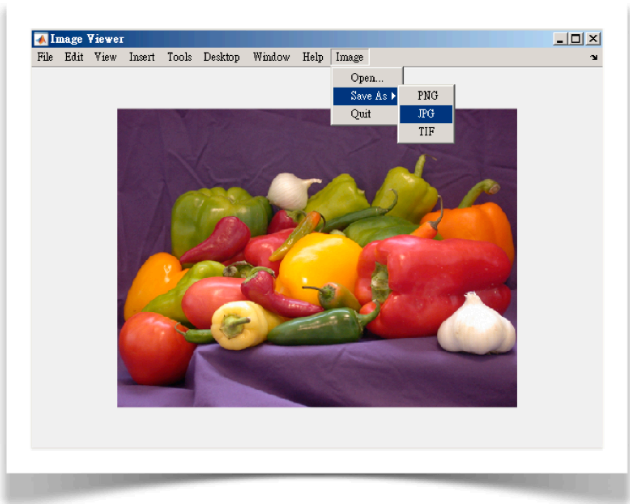


Table 8.4 UI-Menus Properties	
Property	Description
`Label`	Menu label
`Callback`	Callback function when the user select the ui-menu
`Separator`	Separator line mode (off)
`Enable`	Operational state of ui-menu (on)
`Accelerator`	Keyboard equivalent
`Parent`	Parent of ui-menu
Details and More: Help>MATLAB>App Building>GUIDE or Programmatic Workflow>Components and Layout>Properties>Interactive Components>Uimenu Properties	

8.5 Panels, Button Groups, and More UI-Controls

Example08_05.m: Sorting and Searching

[1] This program is a GUI version of Example03_13.m (pages 151-152). The GUI is shown in [5-10], page 312. Note that the functions `sort` and search (lines 134-163) are the same as those in Example03_13.m (page 152).
(Continued at [2], next page.) →

```
 1  sortSearch
 2  
 3  function sortSearch
 4  figure('Position', [30, 30, 400, 400], ...
 5      'Name', 'Sorting and Searching', ...
 6      'MenuBar', 'none', ...
 7      'NumberTitle', 'off', ...
 8      'Resize', 'off')
 9  uicontrol('Style', 'text', ...
10      'String', 'List of Numbers', ...
11      'Units', 'normalized', ...
12      'Position', [.1 .8 .25 .1])
13  hList = uicontrol('Style', 'listbox', ...
14      'Units', 'normalized', ...
15      'Position', [.1 .1 .25 .75]);
16  hPanel1 = uipanel('Position', [.4 .725 .55 .2]);
17      uicontrol(hPanel1, 'Style', 'text', ...
18          'String', 'Enter a Number', ...
19          'Units', 'normalized', ...
20          'Position', [.1 .6 .35 .2])
21      uicontrol(hPanel1, 'Style', 'edit', ...
22          'Callback', @cbEnter, ...
23          'Units', 'normalized', ...
24          'Position', [.1 .1 .35 .4])
25      hSort = uicontrol(hPanel1, 'Style', 'checkbox', ...
26          'String', 'Sort', ...
27          'Callback', @cbSort, ...
28          'Value', true, ...
29          'Units', 'Normalized', ...
30          'Position', [.6 .4 .35 .2]);
31  hPanel2 = uipanel('Position', [.4 .4 .55 .3]);
32      uicontrol(hPanel2, 'Style', 'text', ...
33          'String', 'Find a Number', ...
34          'Units', 'Normalized', ...
35          'Position', [.1 .6 .35 .2])
36      hFind = uicontrol(hPanel2, 'Style', 'edit', ...
37          'Callback', @cbFind, ...
38          'Enable', 'off', ...
39          'Units', 'normalized', ...
40          'Position', [.1 .35 .35 .25]);
```

[2] Example08_05.m (Continued). (Continued at [3], next page.) →

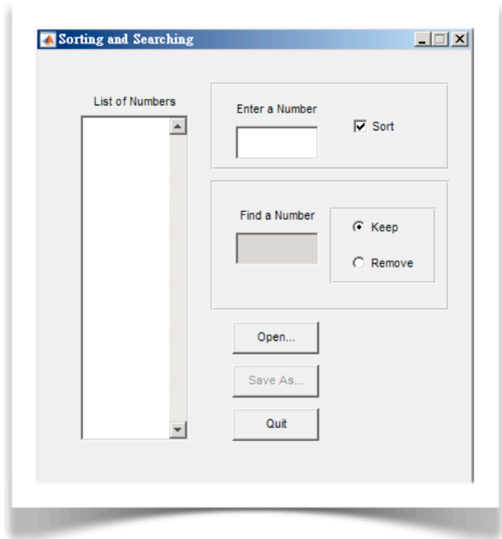
```
    hGroup = uibuttongroup(hPanel2, ...
        'Position', [.5, .2, .45, .6]);
        uicontrol(hGroup, 'Style', 'radiobutton', ...
            'String', 'Keep', ...
            'Value', true, ...
            'Units', 'normalized', ...
            'Position', [.2 .6 .7 .3])
        hRemove = uicontrol(hGroup, 'Style', 'radiobutton', ...
            'String', 'Remove', ...
            'Units', 'normalized', ...
            'Position', [.2 .1 .7 .3]);
uicontrol('Style', 'pushbutton', ...
    'String', 'Open...', ...
    'Callback', @cbOpen, ...
    'Units', 'normalized', ...
    'Position', [.45 .3 .2 .075])
hSaveAs = uicontrol('Style', 'pushbutton', ...
    'String', 'Save As...', ...
    'Callback', @cbSaveAs, ...
    'Enable', 'off', ...
    'Units', 'normalized', ...
    'Position', [.45 .2 .2 .075]);
uicontrol('Style', 'pushbutton', ...
    'String', 'Quit', ...
    'Callback', 'close', ...
    'Units', 'Normalized', ...
    'Position', [.45 .1 .2 .075]);

    function cbEnter(h, ~)
        number = str2double(h.String);
        h.String = [];
        if isempty(hList.String)
            a = [];
        else
            a = str2double(hList.String);
        end
        if search(a, number) > 0
            errordlg('The number exists!')
        else
            a(length(a)+1) = number;
            if hSort.Value
                a = sort(a);
            end
            hList.String = num2cell(a);
        end
        hSaveAs.Enable = 'on';
        hFind.Enable = 'on';
    end

```

[3] Example08_05.m (Continued). (Continued at [4], next page.) →

```
    function cbSort(~, ~)
        a = str2double(hList.String);
        if hSort.Value && ~isempty(a)
            a = sort(a);
            hList.String = num2cell(a);
        end
    end

    function cbFind(h, ~)
        number = str2double(h.String);
        h.String = [];
        a = str2double(hList.String);
        k = search(a, number);
        if k == 0
            errordlg('The number not exist!')
        else
            hList.Value = k;
            if hRemove.Value
                n = length(a);
                b(1:n-1,1) = [a(1:k-1);a(k+1:n)];
                hList.String = num2cell(b);
            end
        end
    end

    function cbOpen(~, ~)
        [file, path] = uigetfile('*.mat');
        if file
            load([path, file], 'a');
            hList.String = num2cell(a);
            hSaveAs.Enable = 'on';
            hFind.Enable = 'on';
        end
    end

    function cbSaveAs(~, ~)
        [file, path] = uiputfile('*.mat');
        if file
            a = str2double(hList.String);
            save([path, file], 'a');
        end
    end
end
```

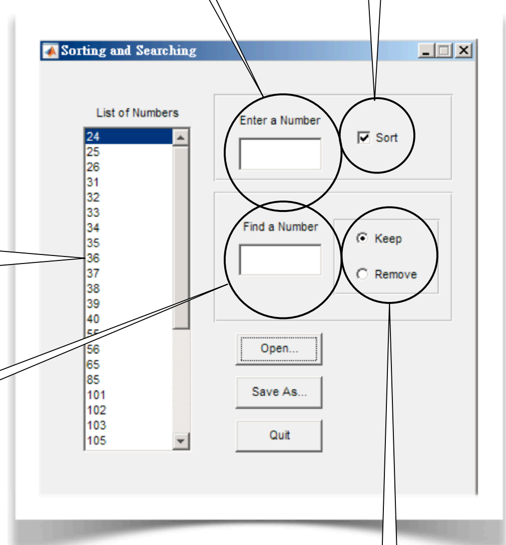
[4] Example08_05.m (Continued).

```
function out = sort(a)
n = length(a);
for i = n-1:-1:1
    for j = 1:i
        if a(j) > a(j+1)
            tmp = a(j);
            a(j) = a(j+1);
            a(j+1) = tmp;
        end
    end
end
out = a;
end

function found = search(a, key)
n = length(a);
low = 1;
high = n;
found = 0;
while low <= high && ~found
    mid = floor((low+high)/2);
    if key == a(mid)
        found = mid;
    elseif key < a(mid)
        high = mid-1;
    else
        low = mid+1;
    end
end
end
```



[6] The user enters numbers in this **editable text box**.

[8] If this **check box** is checked, the list is sorted each time the user enters a number. Otherwise, the number is simply appended to the list.



[7] The numbers show in the **list box**.

[9] The user may search a number by entering a number in this **editable text box**.

[10] States of **radio buttons** within a group are mutually exclusive. When **Keep** is selected, the found number is highlighted. When **Remove** is selected, the found number is removed from the list. →

Table 8.5 UI-Controls and Indicators

Function	Description
`hf = figure`	Create figure window
`ha = axes`	Create axes object
`h = uicontrol(parent,name,value)`	Create UI-control
`h = uitable(parent,name,value)`	Create table UI component
`h = uipanel(parent,name,value)`	Create panel container object
`h = uibuttongroup(parent,name,value)`	Create button group to manage radio buttons and toggle buttons
`h = uitab(parent,name,value)`	Create tabbed panel
`h = uitabgroup(parent,name,value)`	Create container for tabbed panels

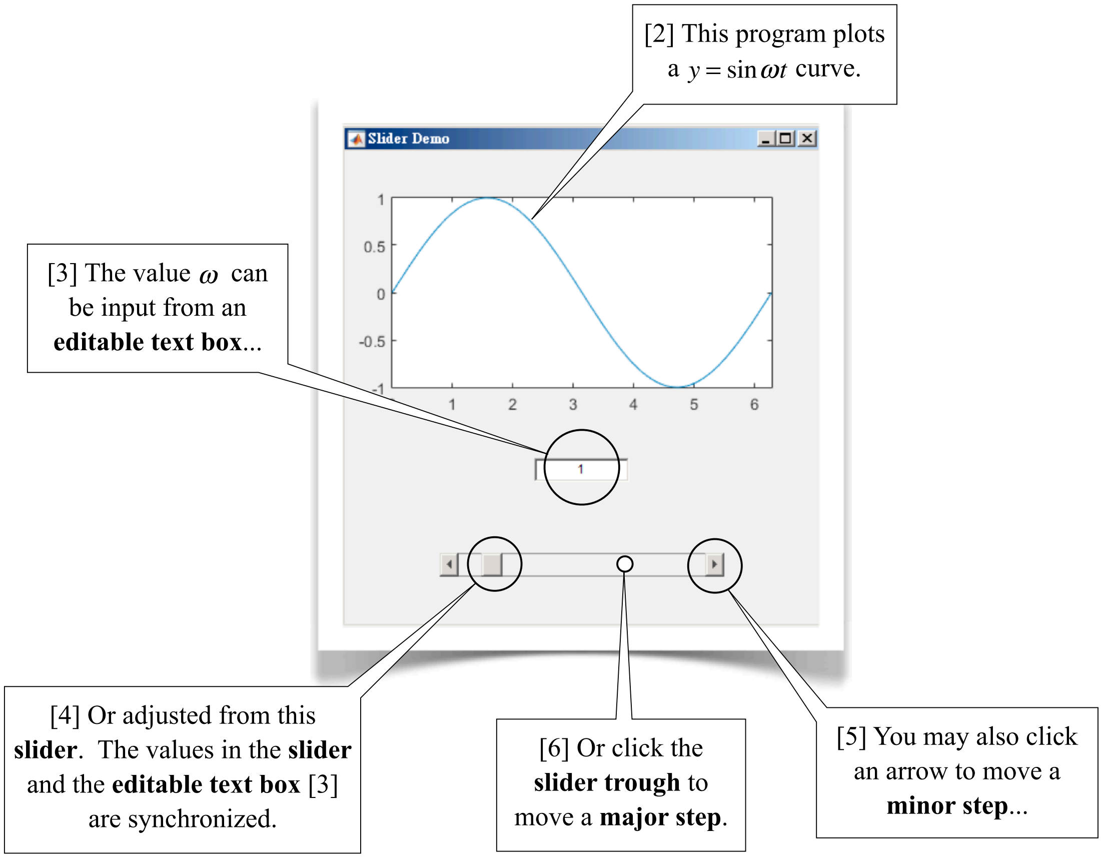
Details and More:
Help>MATLAB>App Building>GUIDE or Programmatic Workflow>Components and Layout>UI Components

8.6 UI-Controls: Sliders

Example08_06.m: Sliders

[1] This program demonstrates the use of a **slider** to adjust the value of an input data (see [2-6], next page). →

```
sliderDemo

function sliderDemo
figure('Position', [50, 50, 400, 400], ...
    'Name', 'Slider Demo', ...
    'MenuBar', 'none', ...
    'NumberTitle', 'off')
axes('Position', [.1 .5 .8 .4])
omega = 1;
sinewave(omega);
hEdit = uicontrol('Style', 'edit', ...
    'String', num2str(omega), ...
    'Callback', @cbEdit, ...
    'Units', 'normalized', ...
    'Position', [.4 .3 .2 .05]);
hSlider = uicontrol('Style', 'slider', ...
    'Callback', @cbSlider, ...
    'Units', 'normalized', ...
    'Position', [.2 .1 .6 .05], ...
    'Value', 1, ...
    'Min', 0, ...
    'Max', 10, ...
    'SliderStep', [0.01, 0.1]);

    function cbEdit(h, ~)
        omega = str2double(h.String);
        hSlider.Value = omega;
        sinewave(omega);
    end

    function cbSlider(h, ~)
        omega = h.Value;
        hEdit.String = num2str(omega);
        sinewave(omega);
    end

    function sinewave(omega)
        t = linspace(0, 2*pi);
        y = sin(omega*t);
        plot(t, y)
        axis([0, 2*pi, -1, 1])
    end
end
```



About Example08_06.m

[7] This program consists of a main program (line 1) and a subfunction `sliderDemo` (lines 3-43). Within the function `sliderDemo`, three nested functions are defined: `cbEdit` (lines 25-29), `cbSlider` (lines 31-35), and `sinewave` (lines 37-42). The function `cbEdit` (lines 25-29) serves as the callback function for the **editable text box** (see [3] and line 13). The function `cbSlider` (lines 31-35) serves as the callback function for the **slider** (see [4-6] and line 17). Both the callback functions use function `sinewave` (lines 37-42), which plots a sine wave (see [2[).

Lines 4-7 create a **Figure** window. Line 8 creates an **Axes** in the **Figure**. Lines 9-10 plot a sine wave with an angular frequency `omega`, which has an initial value of 1 (line 9). Lines 11-15 create an **editable text box** [3]. Lines 16-23 creates a **slider** [4-6]; lines 21-22 specify the range (0-10) for the slider value; line 23 specifies a **minor step** (0.01; also see [5]) and a **major step** (0.1; also see [6]). These step sizes (0.01 and 0.1) are the fraction of the slider range (10).

In the callback function `cbEdit` (lines 25-29), the data in the **editable text box** [3] is retrieved (line 26), the slider value is synchronized (line 27), and the sine wave is plotted (line 28).

In the callback function `cbSlider` (lines 31-35), the slider value [4] is retrieved (line 32), the data in the **editable text box** is synchronized (line 33), and the sine wave is plotted (line 34). #

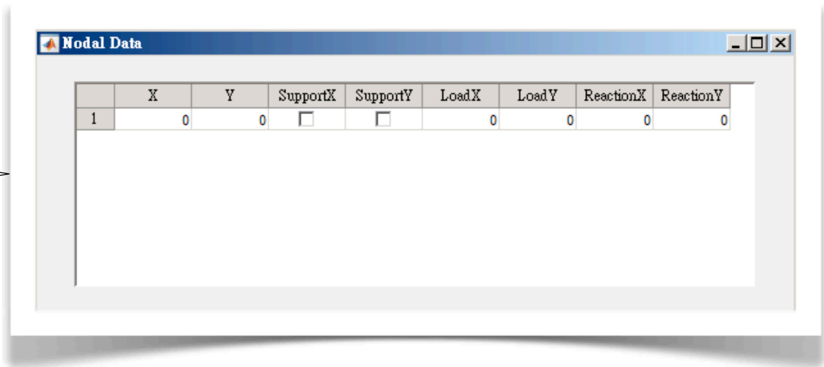
8.7 UI-Tables: Truss Data

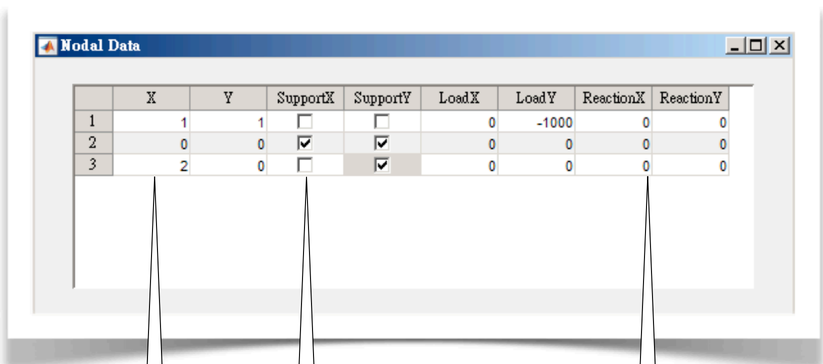
Example08_07a.m: Truss Nodal Data

[1] This program demonstrates the use of a **UI-table** to input the truss nodal data such as those listed in Table3.14a (page 157). Run the program and experience the operations of the **UI-table** as shown in [2-5], next page. This **UI-tables created in this section** will be integrated into the Statically Determinate Trusses program in the next section. →

```
trussNodalData

function trussNodalData
Nodes = struct('x', 0, 'y', 0, ...
    'supportx', false, 'supporty', false, ...
    'loadx', 0, 'loady', 0, ...
    'reactionx', 0, 'reaction', 0);
figure('Position', [30, 30, 590, 200], ...
    'Name', 'Nodal Data', ...
    'MenuBar', 'none', ...
    'NumberTitle', 'off')
data = struct2cell(Nodes)';
columnName = {'X', 'Y', 'SupportX', 'SupportY', ...
    'LoadX', 'LoadY', 'ReactionX', 'ReactionY'};
columnFormat = {'numeric', 'numeric', ...
    'logical', 'logical', ...
    'numeric', 'numeric', 'numeric', 'numeric'};
columnEditable = logical([1 1 1 1 1 1 0 0]);
uitable('Data', data, ...
    'KeyPressFcn', @cbKeyPressNodes, ...
    'ColumnName', columnName, ...
    'ColumnFormat', columnFormat, ...
    'ColumnEditable', columnEditable, ...
    'ColumnWidth', {60}, ...
    'Units', 'normalized', ...
    'Position', [.05 .1 .9 .8]);

    function cbKeyPressNodes(hTable, hKey)
        if strcmpi(hKey.Key, 'downarrow')
            n = size(hTable.Data, 1);
            hTable.Data(n+1,:) = {0 0 false false 0 0 0 0};
        end
    end
end
```

[2] Program Example08_07a.m creates a **UI-table** of eight fields: six numeric and two logical. Initially it has only one row. Each time you press the **down-arrow** key (↓), a row is added to the **UI-table**. Before typing data, click an editable cell and press the **down-arrow** *n*-1 times to add enough rows, where *n* is the number of nodes.





[3] The 1st, 2nd, 5th, and 6th fields are editable numeric fields; you can enter data directly.

[4] The 3rd and 4th fields are logical fields; you simply check (true) or uncheck (false) a box.

[5] The 7th and 8th fields are non-editable numeric fields. They store calculated data.

Example08_07b.m: Truss Member Data

[7] This program uses a **UI-table** to input the truss member data such as those listed in Table3.14b (page 157), as shown in [8]. This program is similar to Example08_07a.m; you should be able to read by yourself.

```
trussMemberData

function trussMemberData
Members = struct('node1', 0, 'node2', 0, 'force', 0);
figure('Position', [30, 30, 260, 200], ...
    'Name', 'Member Data', ...
    'MenuBar', 'none', ...
    'NumberTitle', 'off')
uitable('Data', struct2cell(Members)', ...
    'KeyPressFcn', @cbKeyPressMembers, ...
    'ColumnName', {'Node1', 'Node2', 'Force'}, ...
    'ColumnEditable', logical([1 1 0]), ...
    'ColumnWidth', {60}, ...
    'Units', 'normalized', ...
    'Position', [.05 .1 .9 .8]);

    function cbKeyPressMembers(hTable, hKey)
        if strcmpi(hKey.Key, 'downarrow')
            n = size(hTable.Data, 1);
            hTable.Data(n+1,:) = {0 0 0};
        end
    end
end
```

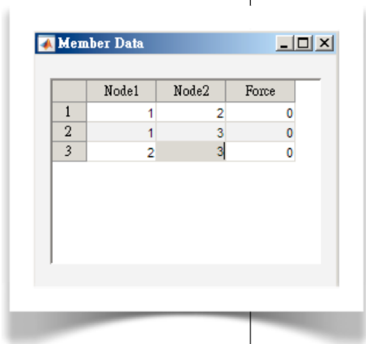


Table 8.7 UI-Table Properties	
Property	Description
`Position`	Location and size of uitable
`Data`	Table content, a cell array
`CellEditCallback`	Cell edit callback function
`KeyPressFcn`	Key press callback function
`ColumnName`	Column heading names
`ColumnFormat`	Cell display format
`ColumnEditable`	Ability to edit column cells
`ColumnWidth`	Width of table columns
`RowName`	Row heading names
`Parent`	Parent of uitable
`FontSize`	Font size

Details and More: Help>MATLAB>App Building>GUIDE or Programmatic Workflow>Components and Layout>Interactive Components>Uitable Properties

8.8 Example: Statically Determinate Trusses (Version 5.0)

Example08_08.m: Truss 5.0

[1] This is a GUI version of program to solve statically determinate truss problems, an improved version of Example06_08.m (pages 272-273). With the GUI, you can input truss data, plot the truss, solve the truss, and so on, in a much more intuitive way, reducing human mistakes (see [6-15], pages 325-326). (Continued at [2], next page.) →

```
trussVersion5

function trussVersion5
Nodes = struct('x', 0, 'y', 0, ...
    'supportx', false, 'supporty', false, ...
    'loadx', 0, 'loady', 0, ...
    'reactionx', 0, 'reactiony', 0);
Members = struct('node1',0, 'node2',0, 'force',0);
hf = figure('Position', [40, 20, 590, 450], ...
    'Name', 'Planar Truss: Untitled', ...
    'MenuBar', 'none', ...
    'NumberTitle', 'off')
axes('Position', [.05 .05 .475 .575]), axis off
data = struct2cell(Nodes)';
columnName = {'X', 'Y', 'SupportX', 'SupportY', ...
    'LoadX', 'LoadY', 'ReactionX', 'ReactionY'};
columnFormat = {'numeric', 'numeric', ...
    'logical', 'logical', ...
    'numeric', 'numeric', 'numeric', 'numeric'};
columnEditable = logical([1 1 1 1 1 1 0 0]);
hNodes = uitable('Data', data, ...
    'KeyPressFcn', @cbKeyPressNodes, ...
    'ColumnName', columnName, ...
    'ColumnFormat', columnFormat, ...
    'ColumnEditable', columnEditable, ...
    'ColumnWidth', {45 45 60 60 60 60 72 72}, ...
    'Units', 'normalized', ...
    'Position', [.05 .65 .9 .275]);
hMembers = uitable('Data', struct2cell(Members)', ...
    'KeyPressFcn', @cbKeyPressMembers, ...
    'ColumnName', {'Node1', 'Node2', 'Force'}, ...
    'ColumnEditable', logical([1 1 0]), ...
    'ColumnWidth', {54, 54, 72}, ...
    'Units', 'normalized', ...
    'Position', [.55 .325 .4 .275]);
```

[2] Example08_08.m (Continued). (Continued at [3], next page.) →

```
uicontrol('Style', 'pushbutton', ...
    'String', 'Plot', ...
    'Callback', @cbPlot, ...
    'Units', 'normalized', ...
    'Position', [.55 .225 .175 .075])
uicontrol('Style', 'pushbutton', ...
    'String', 'Solve', ...
    'Callback', @cbSolve, ...
    'Units', 'normalized', ...
    'Position', [.55 .135 .175 .075])
uicontrol('Style', 'pushbutton', ...
    'String', 'Open', ...
    'Callback', @cbOpen, ...
    'Units', 'normalized', ...
    'Position', [.775 .225 .175 .075])
uicontrol('Style', 'pushbutton', ...
    'String', 'Save As', ...
    'Callback', @cbSaveAs, ...
    'Units', 'normalized', ...
    'Position', [.775 .135 .175 .075])
uicontrol('Style', 'pushbutton', ...
    'String', 'Quit', ...
    'Callback', 'close', ...
    'Units', 'normalized', ...
    'Position', [.775 .05 .175 .075])
uicontrol('Style', 'text', ...
    'String', 'Nodal Data', ...
    'Units', 'normalized', ...
    'Position', [.425 .925 .2 .04])
uicontrol('Style', 'text', ...
    'String', 'Member Data', ...
    'Units', 'normalized', ...
    'Position', [.65 .6 .2 .04])

```

[3] Example08_08.m (Continued). (Continued at [4], next page.) →

```
    function cbPlot(~, ~)
        Nodes = cell2struct(hNodes.Data, fieldnames(Nodes), 2)';
        Members = cell2struct(hMembers.Data, fieldnames(Members), 2)';
        plotTruss(Nodes, Members)
    end

    function cbSolve(~, ~)
        Nodes = cell2struct(hNodes.Data, fieldnames(Nodes), 2)';
        Members = cell2struct(hMembers.Data, fieldnames(Members), 2)';
        [Nodes, Members] = solveTruss(Nodes, Members);
        hNodes.Data = permute(struct2cell(Nodes), [1 3 2])';
        hMembers.Data = permute(struct2cell(Members), [1 3 2])';
    end

    function cbOpen(~, ~)
        [file, path] = uigetfile('*.mat');
        if file
            Nodes = []; Members = [];
            load([path, file])
            hNodes.Data = permute(struct2cell(Nodes), [1 3 2])';
            hMembers.Data = permute(struct2cell(Members), [1 3 2])';
            hf.Name = ['Planar Truss: ', file];
        end
    end

    function cbSaveAs(~, ~)
        [file, path] = uiputfile('*.mat');
        if file
            Nodes = cell2struct(hNodes.Data, fieldnames(Nodes), 2)';
            Members = cell2struct(hMembers.Data, fieldnames(Members), 2)';
            save([path, file], 'Nodes', 'Members')
            hf.Name = ['Planar Truss: ', file];
        end
    end

    function cbKeyPressNodes(hTable, hKey)
        if strcmpi(hKey.Key, 'downarrow')
            n = size(hTable.Data, 1);
            hTable.Data(n+1,:) = {0 0 false false 0 0 0 0};
        end
    end

    function cbKeyPressMembers(hTable, hKey)
        if strcmpi(hKey.Key, 'downarrow')
            n = size(hTable.Data, 1);
            hTable.Data(n+1,:) = {0 0 0};
        end
    end
end
```

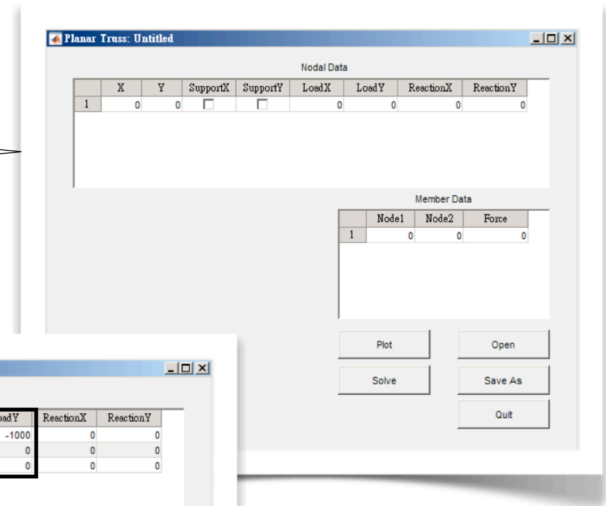
[4] Example08_08.m (Continued). (Continued at [5], next page.) →

```
function [outNodes, outMembers] = solveTruss(Nodes, Members)
n = size(Nodes,2); m = size(Members,2);
if (m+3) < 2*n
    disp('Unstable!')
    outNodes = 0; outMembers = 0; return
elseif (m+3) > 2*n
    disp('Statically indeterminate!')
    outNodes = 0; outMembers = 0; return
end
A = zeros(2*n, 2*n); loads = zeros(2*n,1); nsupport = 0;
for i = 1:n
    for j = 1:m
        if Members(j).node1 == i || Members(j).node2 == i
            if Members(j).node1 == i
                n1 = i; n2 = Members(j).node2;
            elseif Members(j).node2 == i
                n1 = i; n2 = Members(j).node1;
            end
            x1 = Nodes(n1).x; y1 = Nodes(n1).y;
            x2 = Nodes(n2).x; y2 = Nodes(n2).y;
            L = sqrt((x2-x1)^2+(y2-y1)^2);
            A(2*i-1,j) = (x2-x1)/L;
            A(2*i,  j) = (y2-y1)/L;
        end
    end
    if (Nodes(i).supportx == 1)
        nsupport = nsupport+1;
        A(2*i-1,m+nsupport) = 1;
    end
    if (Nodes(i).supporty == 1)
        nsupport = nsupport+1;
        A(2*i, m+nsupport) = 1;
    end
    loads(2*i-1) = -Nodes(i).loadx;
    loads(2*i)   = -Nodes(i).loady;
end
forces = A\loads;
for j = 1:m
    Members(j).force = forces(j);
end
nsupport = 0;
for i = 1:n
    Nodes(i).reactionx = 0;
    Nodes(i).reactiony = 0;
    if (Nodes(i).supportx == 1)
        nsupport = nsupport+1;
        Nodes(i).reactionx = forces(m+nsupport);
    end
    if (Nodes(i).supporty == 1)
        nsupport = nsupport+1;
        Nodes(i).reactiony = forces(m+nsupport);
    end
end
outNodes = Nodes; outMembers = Members;
disp('Solved successfully.')
end

```

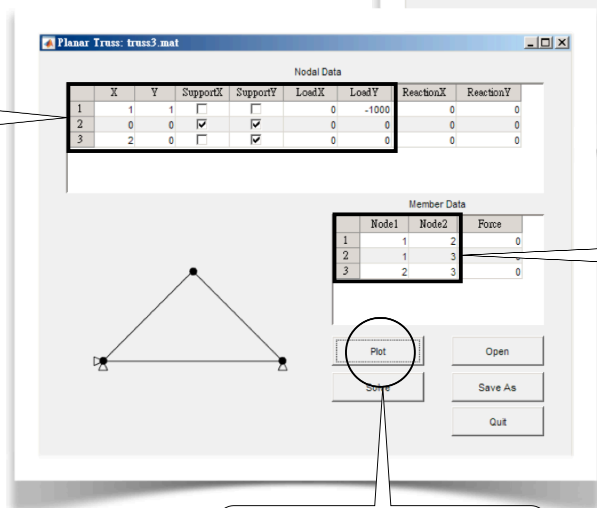
[5] Example08_08.m (Continued). →

```
function plotTruss(Nodes, Members)
if (size(fieldnames(Nodes),1)<6 || size(fieldnames(Members),1)<2)
    disp('Truss data not complete'); return
end
n = length(Nodes); m = length(Members);
minX = Nodes(1).x; maxX = Nodes(1).x;
minY = Nodes(1).y; maxY = Nodes(1).y;
for k = 2:n
    if (Nodes(k).x < minX) minX = Nodes(k).x; end
    if (Nodes(k).x > maxX) maxX = Nodes(k).x; end
    if (Nodes(k).y < minY) minY = Nodes(k).y; end
    if (Nodes(k).y > maxY) maxY = Nodes(k).y; end
end
rangeX = maxX-minX; rangeY = maxY-minY;
axis([minX-rangeX/5, maxX+rangeX/5, minY-rangeY/5, maxY+rangeY/5])
ha = gca; delete(ha.Children)
axis equal off
hold on
for k = 1:m
    n1 = Members(k).node1; n2 = Members(k).node2;
    x = [Nodes(n1).x, Nodes(n2).x];
    y = [Nodes(n1).y, Nodes(n2).y];
    plot(x,y,'k-o', 'MarkerFaceColor', 'k')
end
for k = 1:n
    if Nodes(k).supportx
        x = [Nodes(k).x, Nodes(k).x-rangeX/20, Nodes(k).x-rangeX/20, Nodes(k).x];
        y = [Nodes(k).y, Nodes(k).y+rangeX/40, Nodes(k).y-rangeX/40, Nodes(k).y];
        plot(x,y,'k-')
    end
    if Nodes(k).supporty
        x = [Nodes(k).x, Nodes(k).x-rangeX/40, Nodes(k).x+rangeX/40, Nodes(k).x];
        y = [Nodes(k).y, Nodes(k).y-rangeX/20, Nodes(k).y-rangeX/20, Nodes(k).y];
        plot(x,y,'k-')
    end
end
end
```

[6] Run the program. This is the initial GUI, which has two UI-tables (**Nodal Data** and **Member Data**), 5 pushbuttons (**Plot**, **Solve**, **Open**, **Save As**, and **Quit**), and an invisible **axes**.



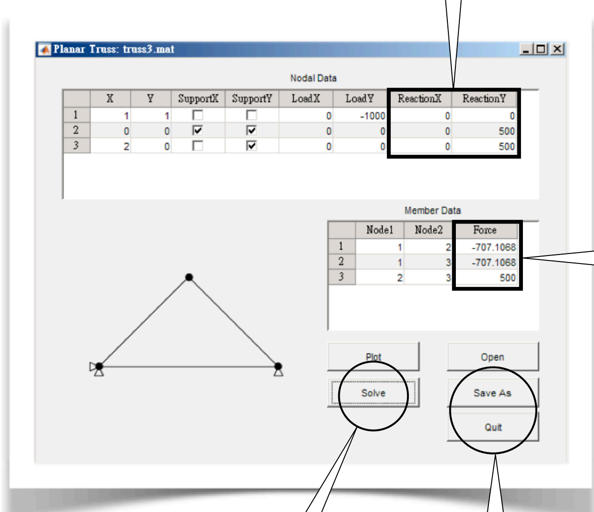
[7] Nodal data are input here (see 8.7[2-5], page 318). Remember, before typing the data, click an editable cell and press the **down-arrow** n-1 times to add enough rows, where n is the number of nodes.



[8] Member data are input here (see 8.7[8], page 319). Remember, before typing the data, click an editable cell and press the **down-arrow** m-1 times to add enough rows, where m is the number of truss members.

[9] Click **Plot** button to plot the truss on the **axes**.

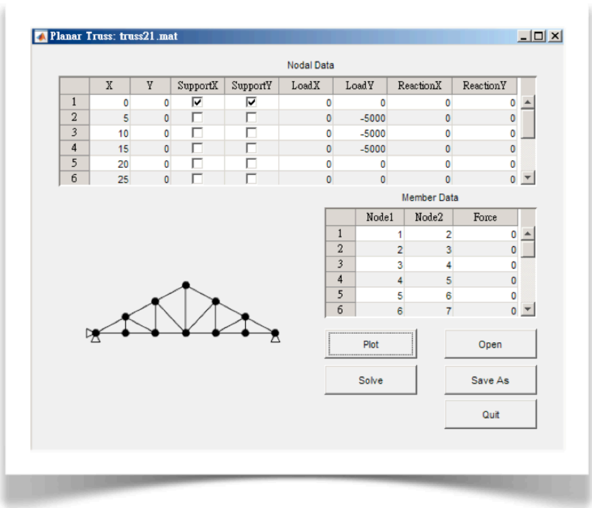
[11] The reaction forces are displayed here.

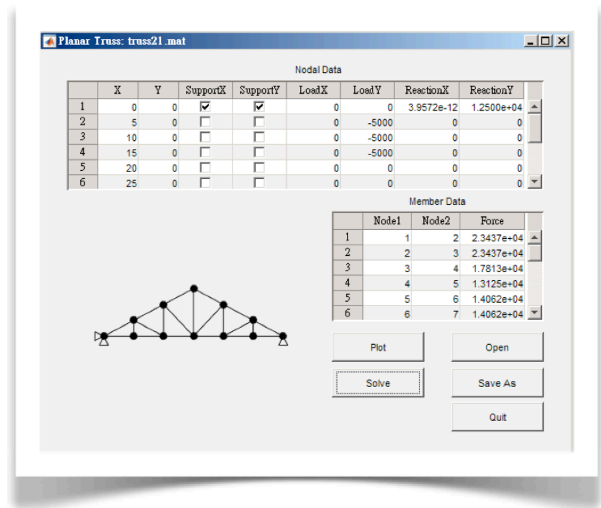


[12] The member forces are displayed here.

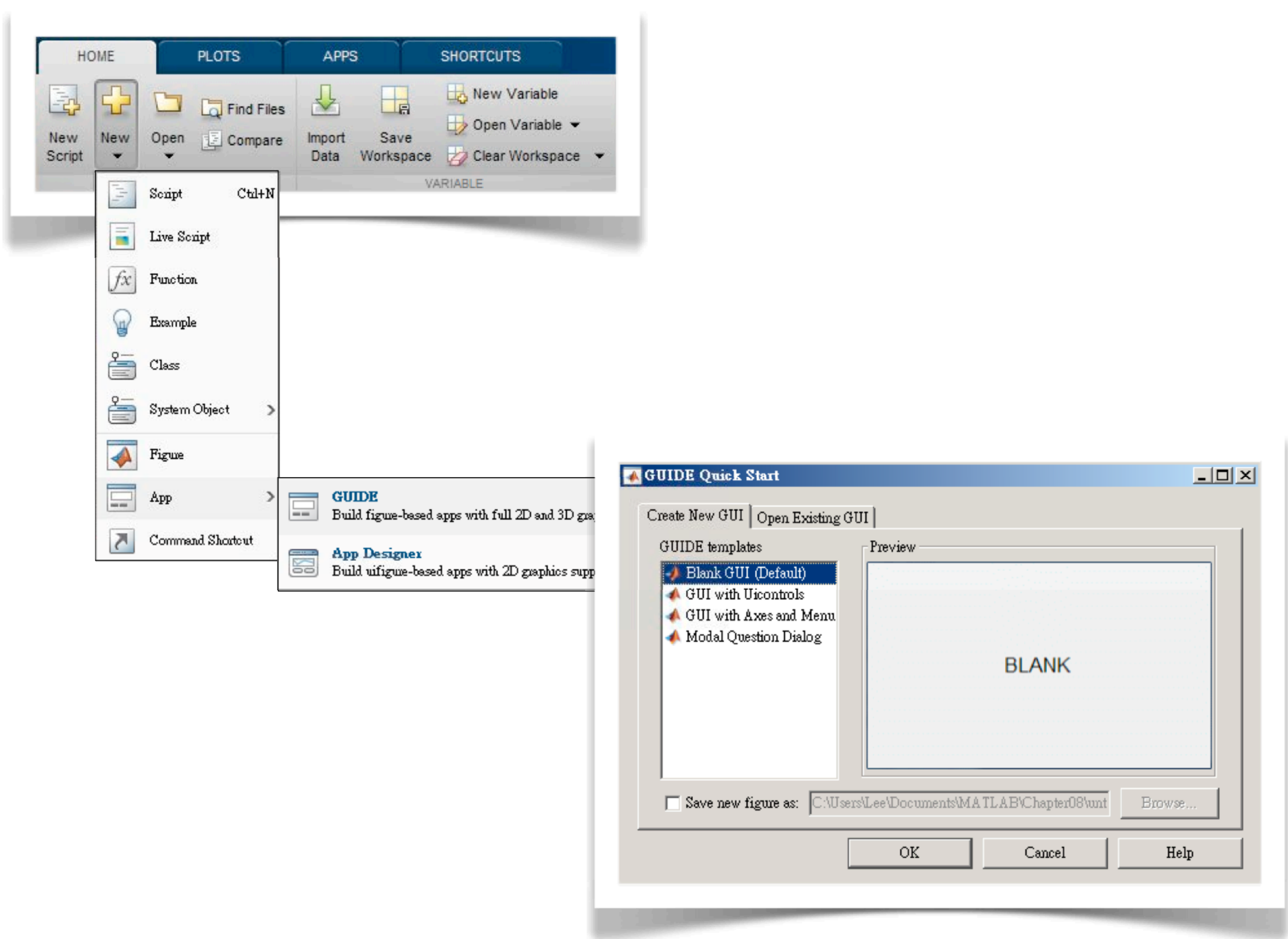
[10] Click **Solve** button.

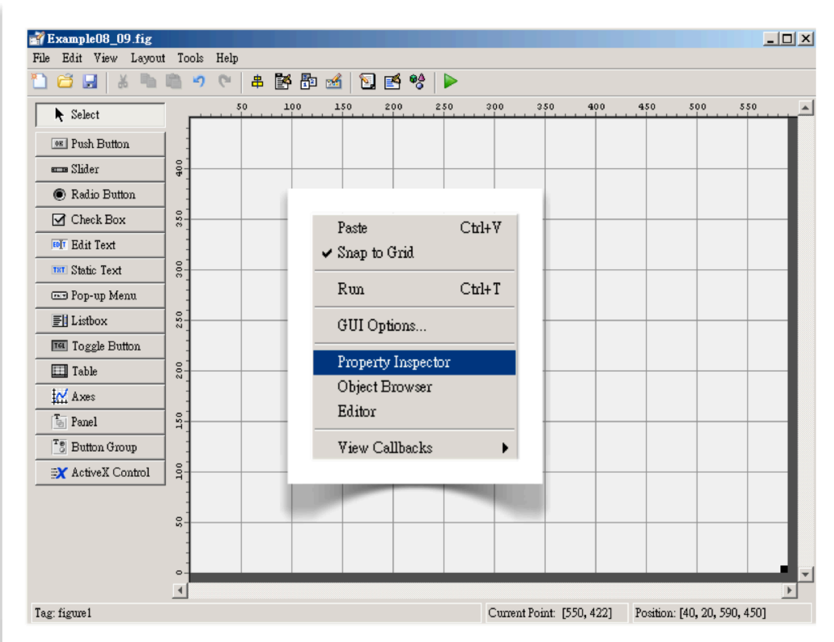
[13] Save the file and quit the program. →

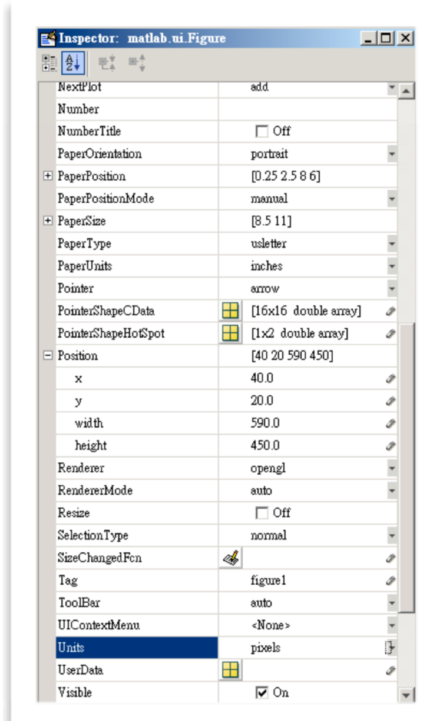

Planar Truss: truss21.mat
Nodal Data
X Y SupportX SupportY LoadX LoadY ReactionX ReactionY
1 0 0 0 0 0 0
2 5 0 0 -5000 0 0
3 10 0 0 -5000 0 0
4 15 0 0 -5000 0 0
5 20 0 0 0 0 0
6 25 0 0 0 0 0
Member Data
Node1 Node2 Force
1 1 2 0
2 2 3 0
3 3 4 0
4 4 5 0
5 5 6 0
6 6 7 0
Plot
Open
Solve
Save As
Quit

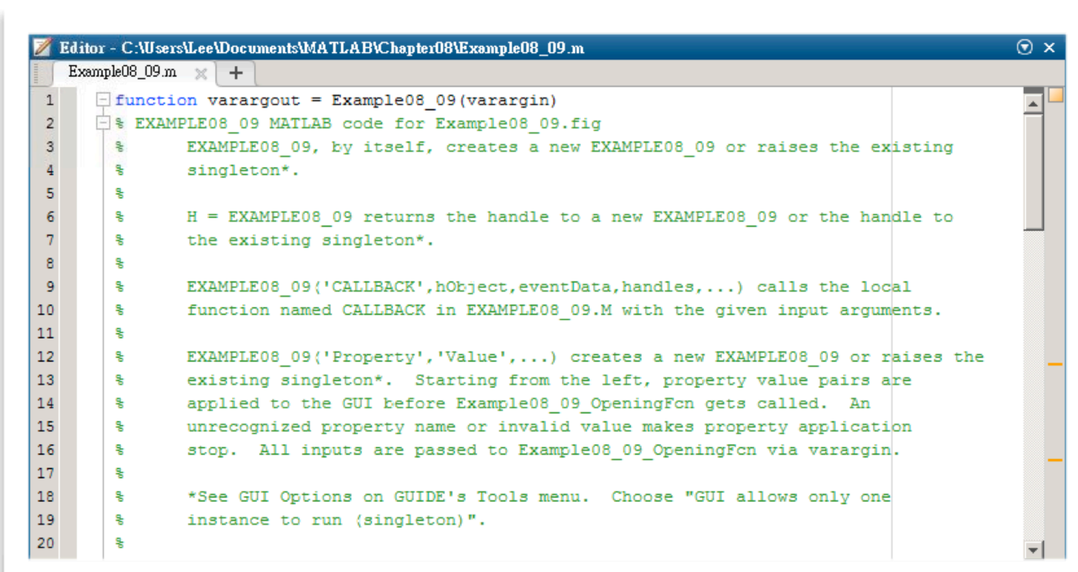

Planar Truss: truss21.mat
Nodal Data
X Y SupportX SupportY LoadX LoadY ReactionX ReactionY
1 0 0 0 0 3.9572e-12 1.2500e+04
2 5 0 0 -5000 0 0
3 10 0 0 -5000 0 0
4 15 0 0 -5000 0 0
5 20 0 0 0 0 0
6 25 0 0 0 0 0
Member Data
Node1 Node2 Force
1 1 2 2.3437e+04
2 2 3 2.3437e+04
3 3 4 1.7813e+04
4 4 5 1.3125e+04
5 5 6 1.4062e+04
6 6 7 1.4062e+04
Plot
Open
Solve
Save As
Quit

8.9 GUIDE: Graphical User Interface Development Environment

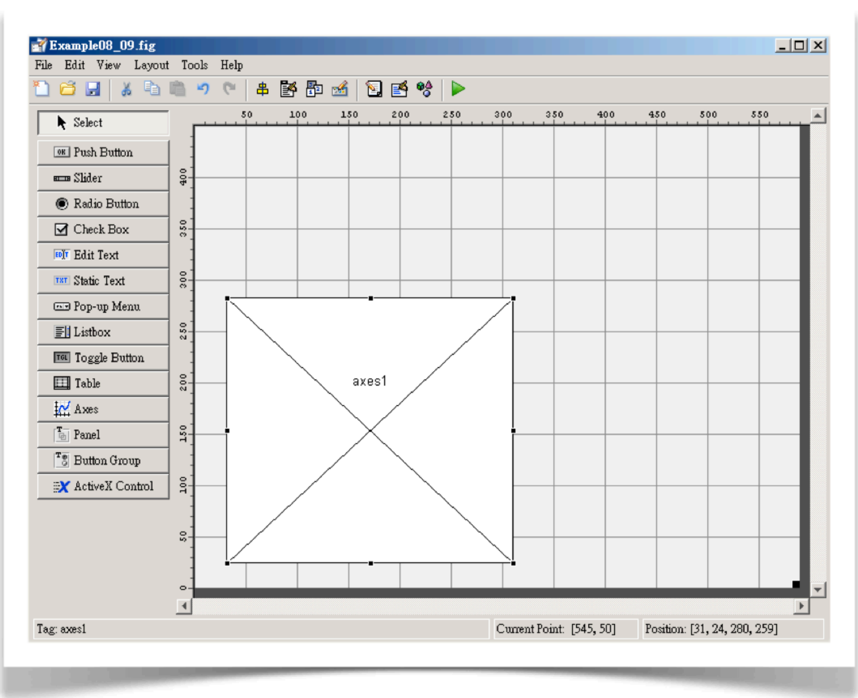



Example08_09.fig
File Edit View Layout Tools Help
Select
Push Button
Slider
Radio Button
Check Box
Edit Text
Static Text
Pop-up Menu
Listbox
Toggle Button
Table
Axes
Panel
Button Group
ActiveX Control
Paste Ctrl+V
Snap to Grid
Run Ctrl+T
GUI Options...
Property Inspector
Object Browser
Editor
View Callbacks
Tag: figure1
Current Point: [550, 422]
Position: [40, 20, 590, 450]

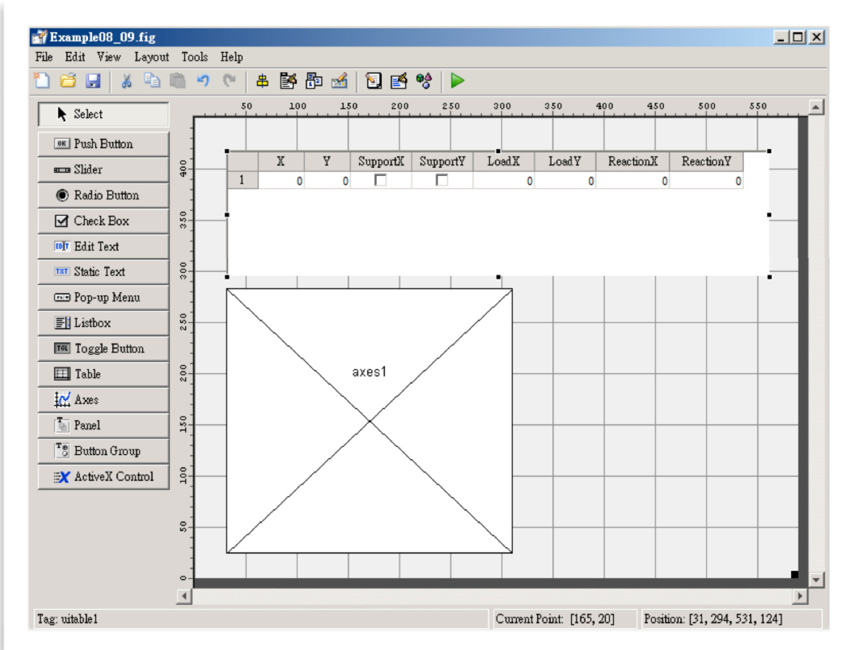

Inspector: matlab.ui.Figure
NextPlot add
Number
NumberTitle Off
PaperOrientation portrait
PaperPosition [0.25 2.5 8 6]
PaperPositionMode manual
PaperSize [8.5 11]
PaperType usletter
PaperUnits inches
Pointer arrow
PointerShapeCData [16x16 double array]
PointerShapeHotSpot [1x2 double array]
Position [40 20 590 450]
x 40.0
y 20.0
width 590.0
height 450.0
Renderer opengl
RendererMode auto
Resize Off
SelectionType normal
SizeChangedFcn
Tag figure1
ToolBar auto
UIContextMenu <None>
Units pixels
UserData
Visible On

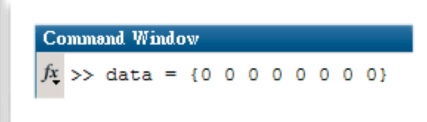

Editor - C:\Users\Lee\Documents\MATLAB\Chapter08\Example08_09.m
Example08_09.m
function varargout = Example08_09(varargin)
% EXAMPLE08_09 MATLAB code for Example08_09.fig
% EXAMPLE08_09, by itself, creates a new EXAMPLE08_09 or raises the existing
% singleton*.
%
% H = EXAMPLE08_09 returns the handle to a new EXAMPLE08_09 or the handle to
% the existing singleton*.
%
% EXAMPLE08_09('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in EXAMPLE08_09.M with the given input arguments.
%
% EXAMPLE08_09('Property','Value',...) creates a new EXAMPLE08_09 or raises the
% existing singleton*. Starting from the left, property value pairs are
% applied to the GUI before Example08_09_OpeningFcn gets called. An
% unrecognized property name or invalid value makes property application
% stop. All inputs are passed to Example08_09_OpeningFcn via varargin.
%
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)".
%


```
global hf Nodes Members
hf = hObject;
Nodes = struct('x', 0, 'y', 0, ...
    'supportx', false, 'supporty', false, ...
    'loadx', 0, 'loady', 0, ...
    'reactionx', 0, 'reactiony', 0);
Members = struct('node1', 0, 'node2', 0, 'force', 0);
```




```
axis off
```


Example08_09.fig
File Edit View Layout Tools Help
Select
Push Button
Slider
Radio Button
Check Box
Edit Text
Static Text
Pop-up Menu
Listbox
Toggle Button
Table
Axes
Panel
Button Group
ActiveX Control
X Y SupportX SupportY LoadX LoadY ReactionX ReactionY
axes1
Tag: uitable1
Current Point: [165, 20]
Position: [31, 294, 531, 124]


Command Window
>> data = {0 0 0 0 0 0 0 0}


Data
[4x2 cell array]

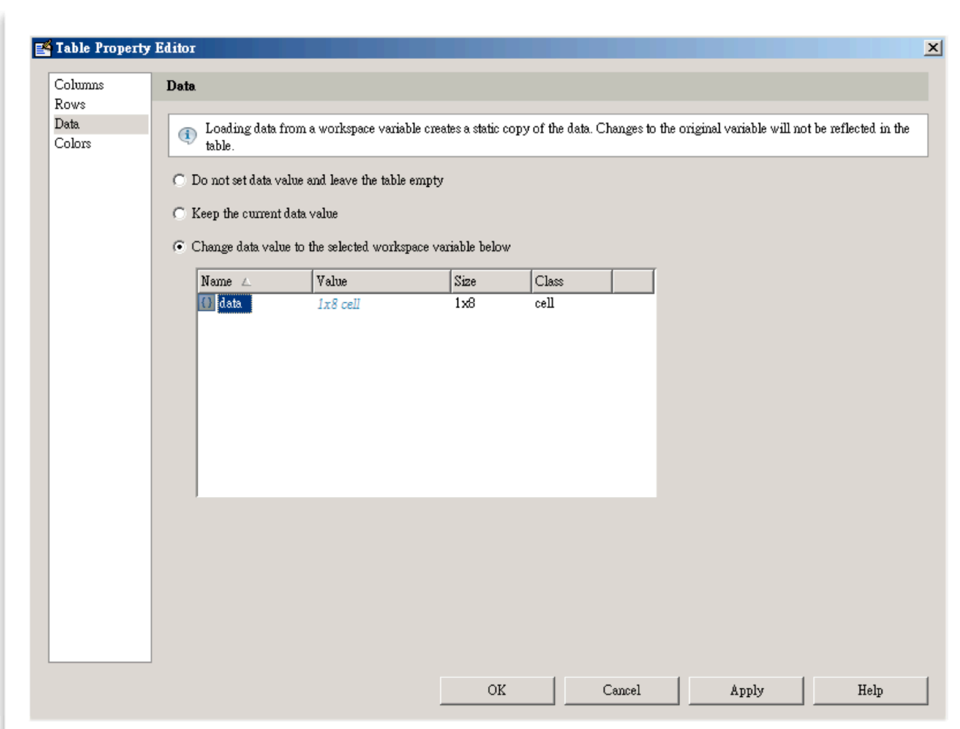
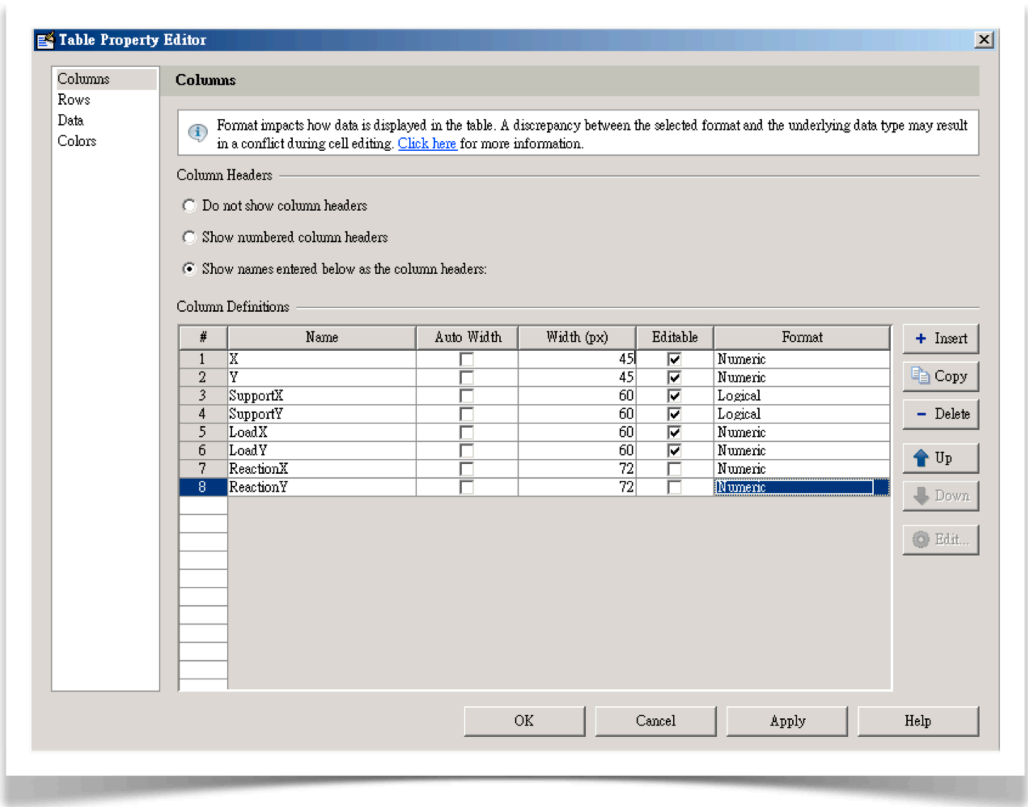
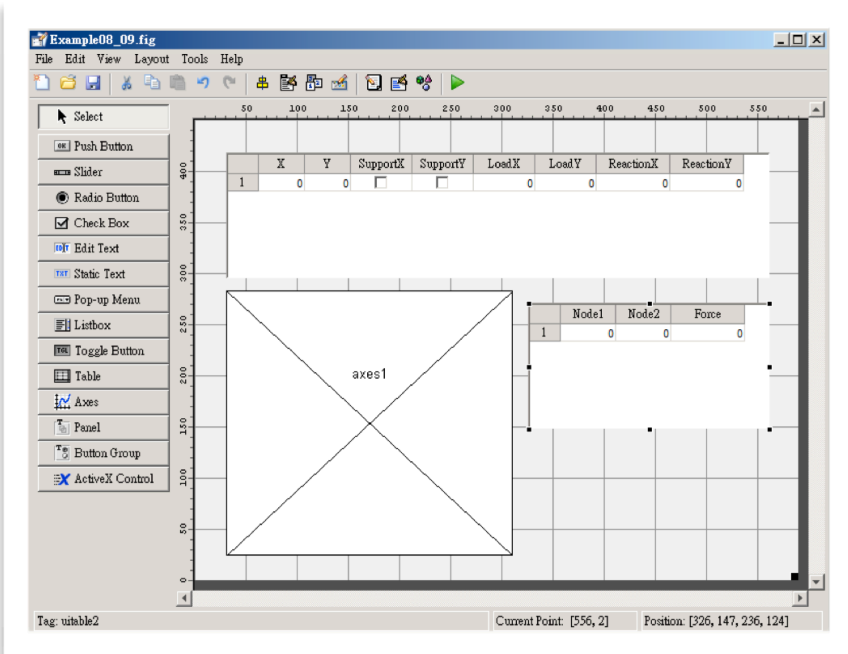


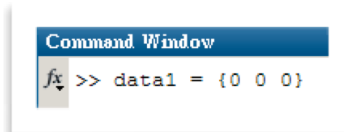
Table Property Editor
Columns
Rows
Data
Colors
Data
Loading data from a workspace variable creates a static copy of the data. Changes to the original variable will not be reflected in the table.
Do not set data value and leave the table empty
Keep the current data value
Change data value to the selected workspace variable below
Name Value Size Class
data 1x8 cell 1x8 cell
OK
Cancel
Apply
Help




```
global hNodes
hNodes = hObject;
```


```
if strcmpi(eventdata.Key, 'downarrow')
    n = size(hObject.Data, 1);
    hObject.Data(n+1,:) = {0 0 false false 0 0 0 0};
end
```


Example08_09.fig
File Edit View Layout Tools Help
Select
Push Button
Slider
Radio Button
Check Box
Edit Text
Static Text
Pop-up Menu
Listbox
Toggle Button
Table
Axes
Panel
Button Group
ActiveX Control
X Y SupportX SupportY LoadX LoadY ReactionX ReactionY
axes1
Node1 Node2 Force
Tag: uitable2
Current Point: [556, 2]
Position: [326, 147, 236, 124]


Command Window
>> data1 = {0 0 0}


Data
[4x2 cell array]

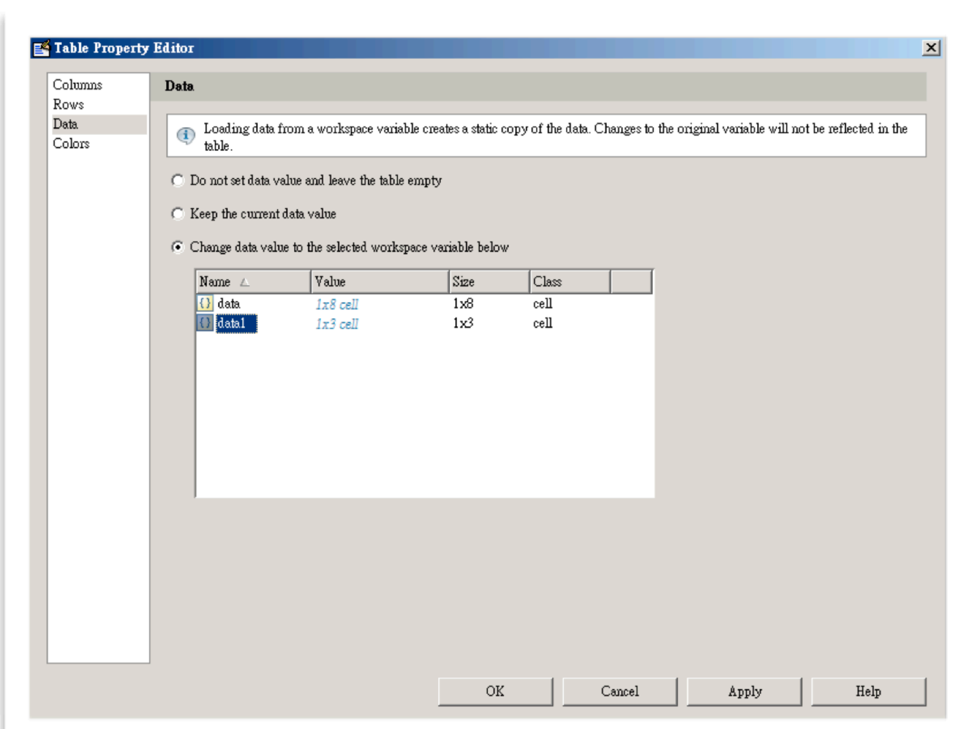
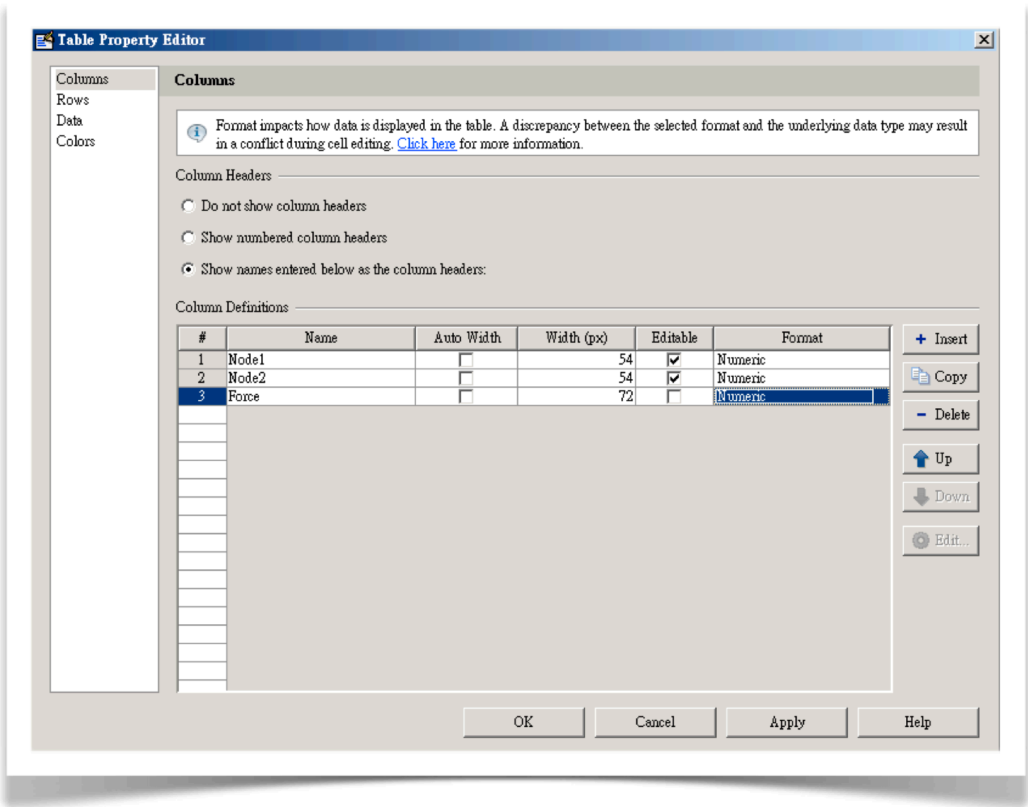
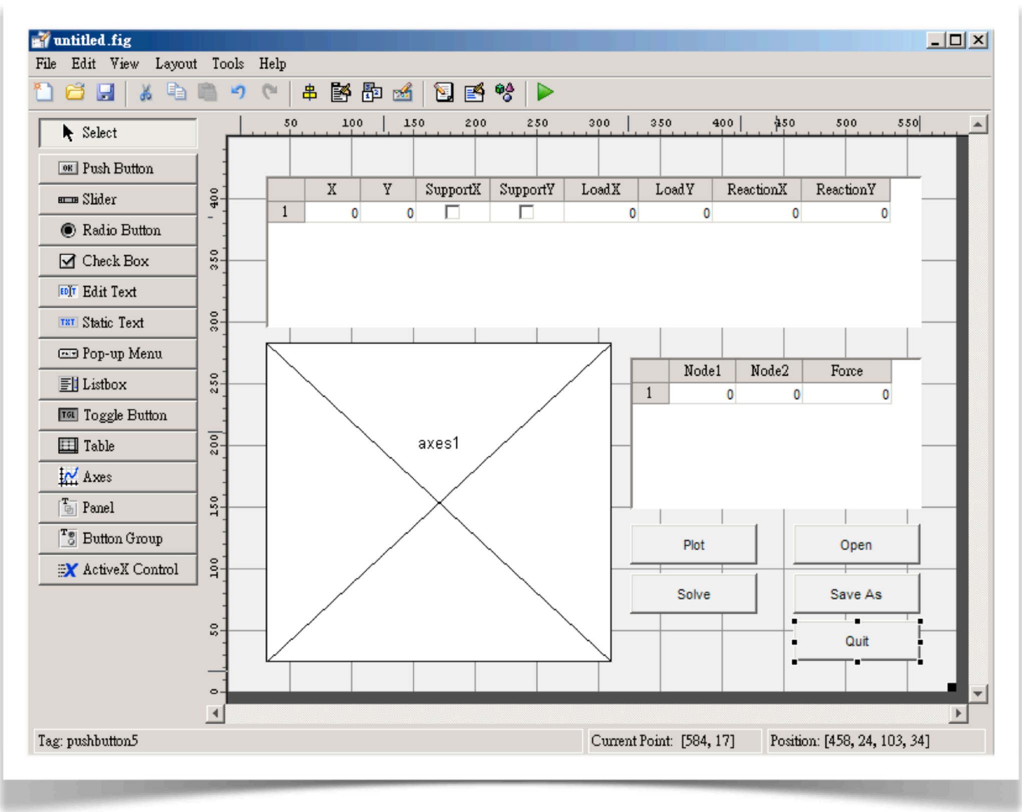


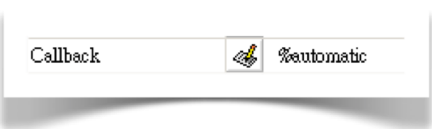
Table Property Editor
Columns
Rows
Data
Colors
Data
Loading data from a workspace variable creates a static copy of the data. Changes to the original variable will not be reflected in the table.
Do not set data value and leave the table empty
Keep the current data value
Change data value to the selected workspace variable below
Name Value Size Class
data 1x8 cell 1x8 cell
data1 1x3 cell 1x3 cell
OK
Cancel
Apply
Help




```
global hMembers
hMembers = hObject;
```


```
if strcmpi(eventdata.Key, 'downarrow')
    n = size(hObject.Data, 1);
    hObject.Data(n+1,:) = {0 0 0};
end
```



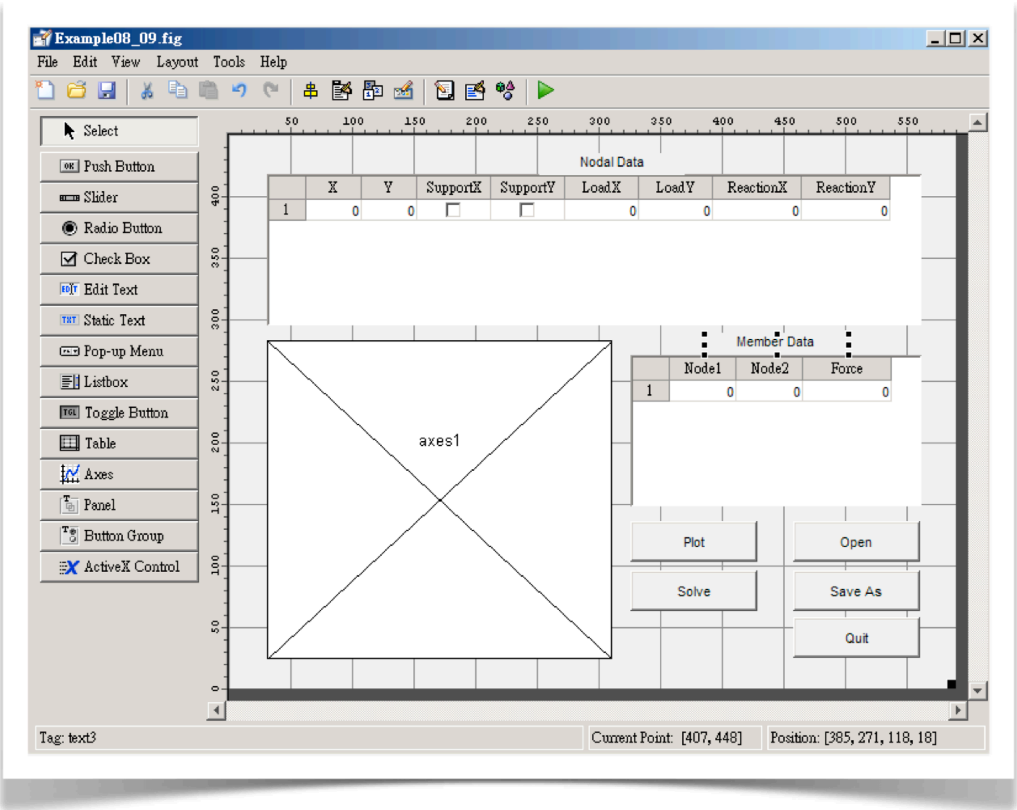


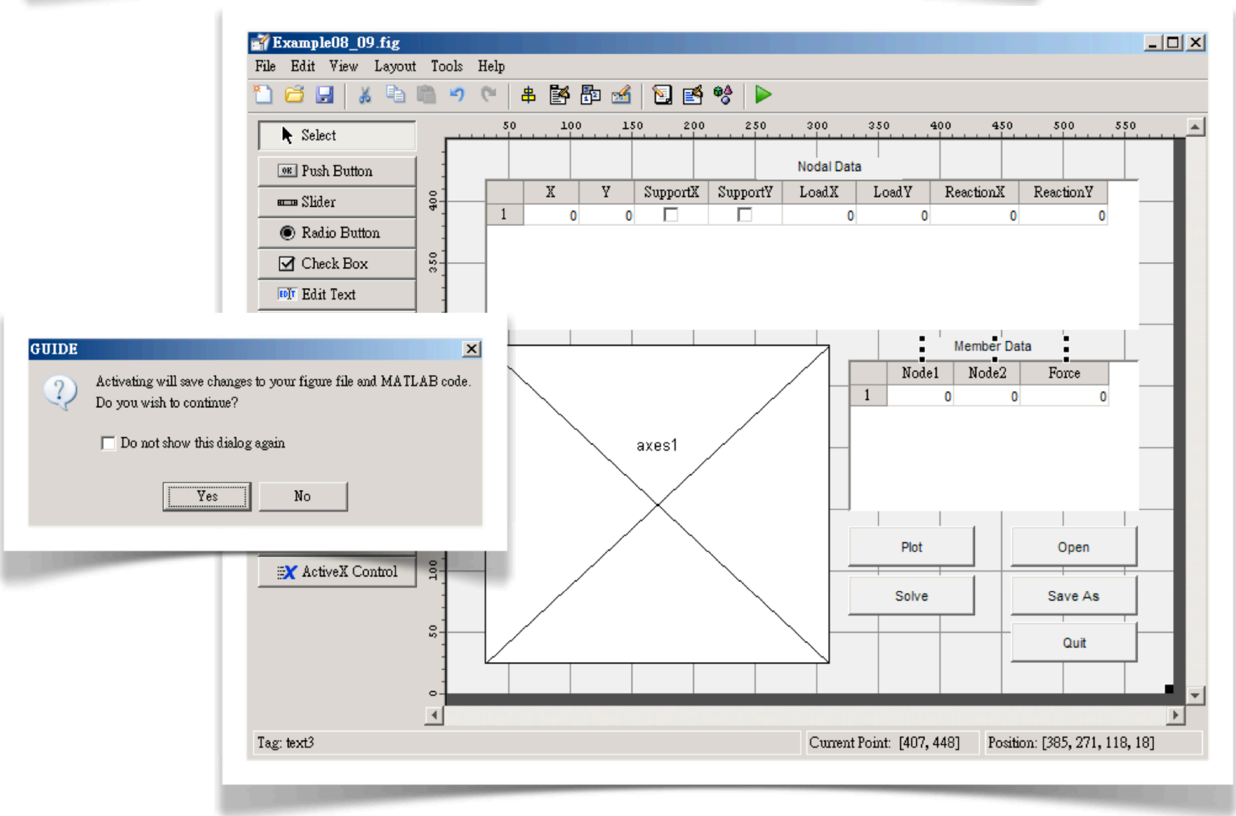

```
global Nodes Members hNodes hMembers
Nodes = cell2struct(hNodes.Data, fieldnames(Nodes), 2)';
Members = cell2struct(hMembers.Data, fieldnames(Members), 2)';
plotTruss(Nodes, Members)
```

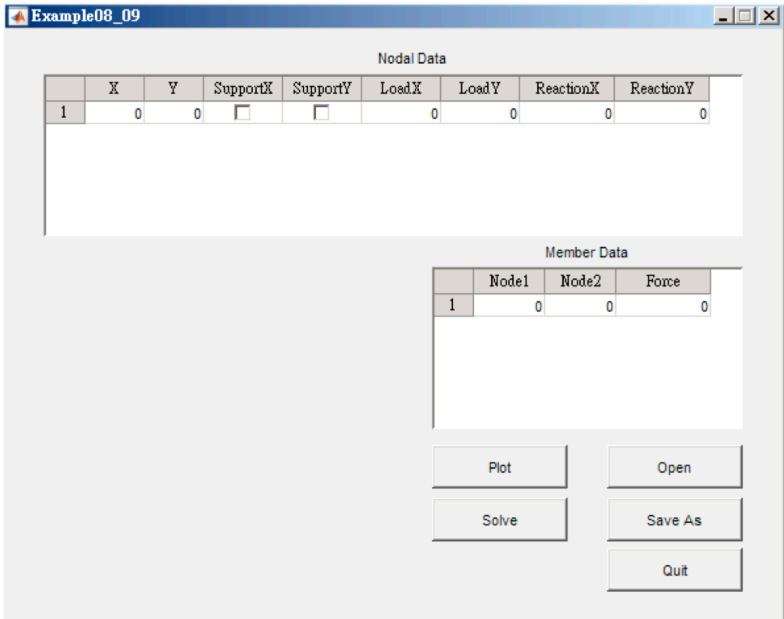
```
global Nodes Members hNodes hMembers
Nodes = cell2struct(hNodes.Data, fieldnames(Nodes), 2)';
Members = cell2struct(hMembers.Data, fieldnames(Members), 2)';
[Nodes, Members] = solveTruss(Nodes, Members);
hNodes.Data = permute(struct2cell(Nodes), [1 3 2])';
hMembers.Data = permute(struct2cell(Members), [1 3 2])';
```

```
global Nodes Members hNodes hMembers hf
[file, path] = uigetfile('*.mat');
if file
    Nodes = []; Members = [];
    load([path, file])
    hNodes.Data = permute(struct2cell(Nodes), [1 3 2])';
    hMembers.Data = permute(struct2cell(Members), [1 3 2])';
    hf.Name = ['Planar Truss: ', file];
end
```

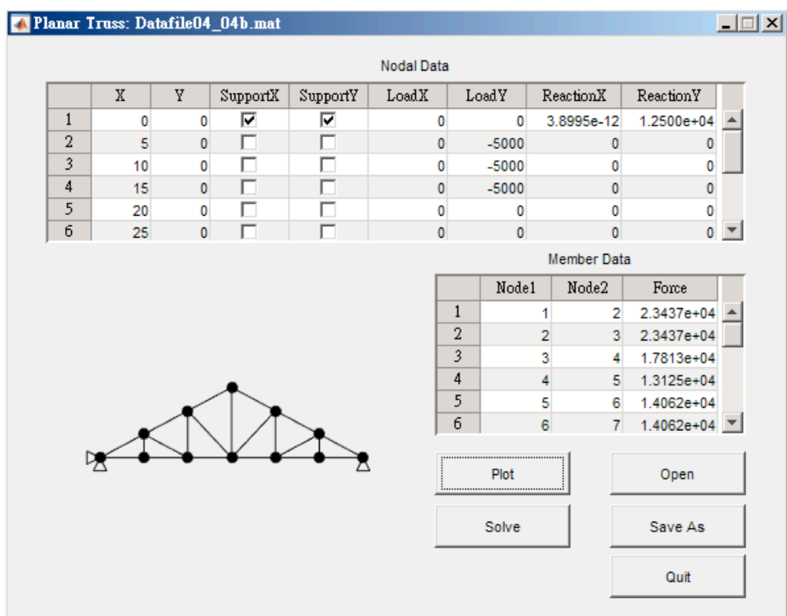
```
global Nodes Members hNodes hMembers hf
[file, path] = uiputfile('*.mat');
if file
    Nodes = cell2struct(hNodes.Data, fieldnames(Nodes), 2)';
    Members = cell2struct(hMembers.Data, fieldnames(Members), 2)';
    save([path, file], 'Nodes', 'Members')
    hf.Name = ['Planar Truss: ', file];
end
```


Example08_09.fig
File Edit View Layout Tools Help
Select
Push Button
Slider
Radio Button
Check Box
Edit Text
Static Text
Pop-up Menu
Listbox
Toggle Button
Table
Axes
Panel
Button Group
ActiveX Control
Nodal Data
X Y SupportX SupportY LoadX LoadY ReactionX ReactionY
Member Data
Node1 Node2 Force
axes1
Plot
Open
Solve
Save As
Quit
Tag: text3
Current Point: [407, 448]
Position: [385, 271, 118, 18]

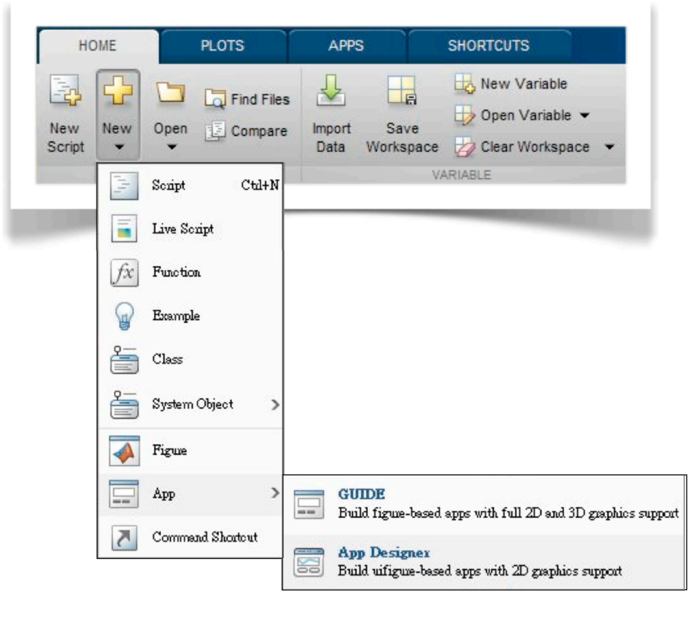

Example08_09.fig
GUIDE
Activating will save changes to your figure file and MATLAB code.
Do you wish to continue?
Do not show this dialog again
Yes
No
axes1
Plot
Open
Solve
Save As
Quit
Tag: text3
Current Point: [407, 448]
Position: [385, 271, 118, 18]

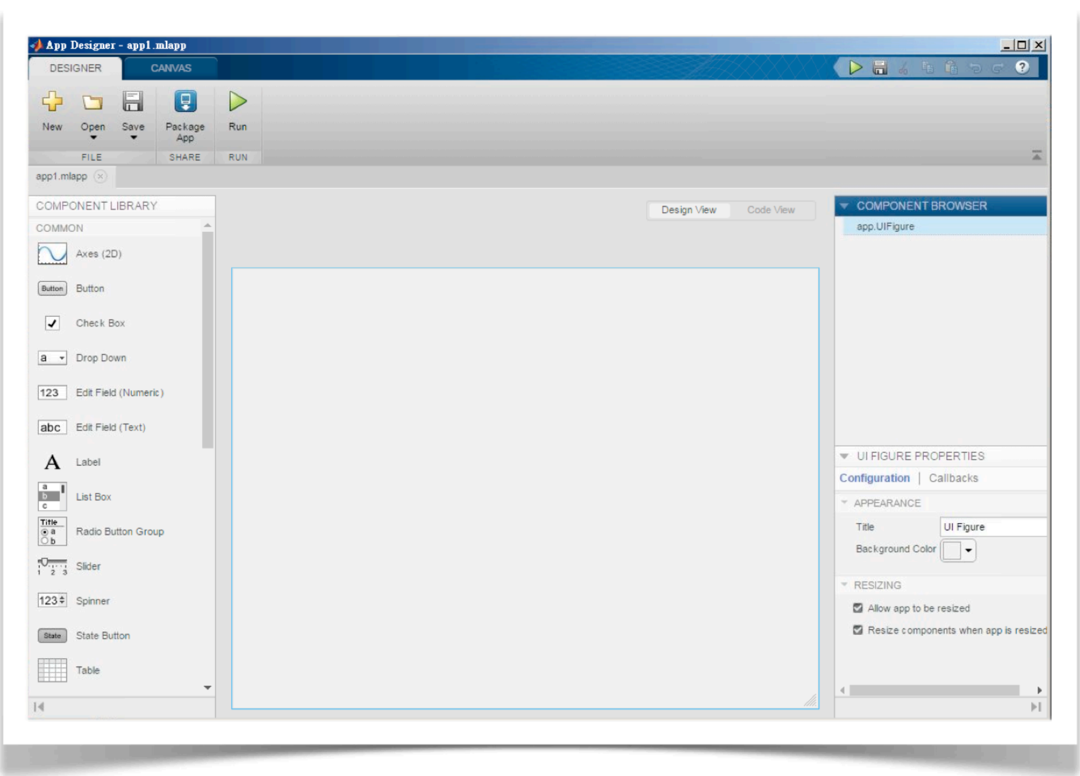

Example08_09
Nodal Data
X
Y
SupportX
SupportY
LoadX
LoadY
ReactionX
ReactionY
Member Data
Node1
Node2
Force
Plot
Open
Solve
Save As
Quit

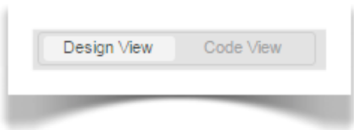

EDITOR
PUBLISH
Breakpoints
Run
BREAKPOINTS

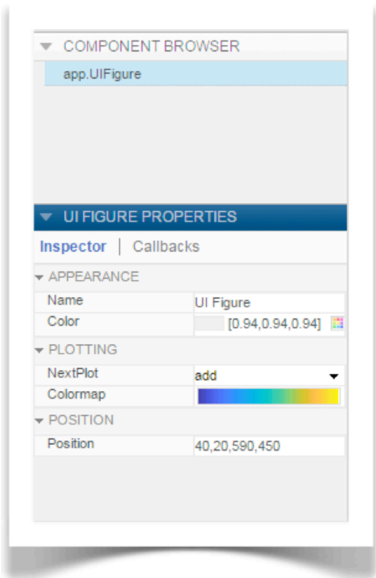

Planar Truss: Datafile04_04b.mat
Nodal Data
X
Y
SupportX
SupportY
LoadX
LoadY
ReactionX
ReactionY
Member Data
Node1
Node2
Force
Plot
Open
Solve
Save As
Quit

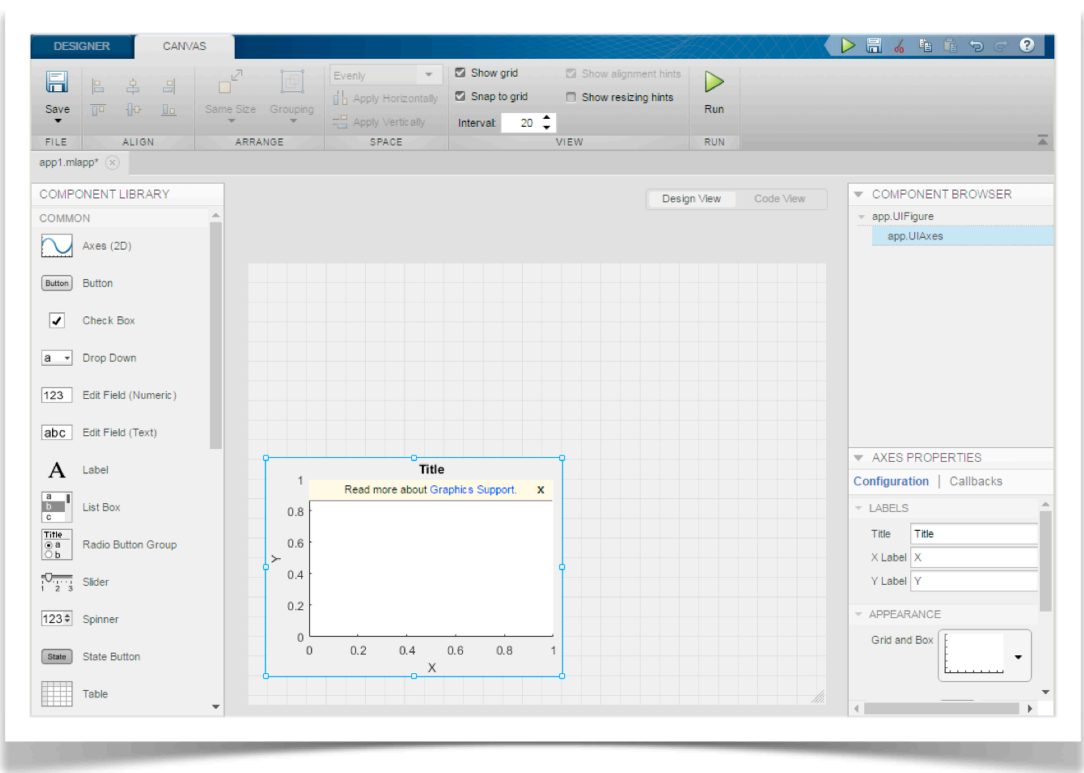
8.10 App Designer

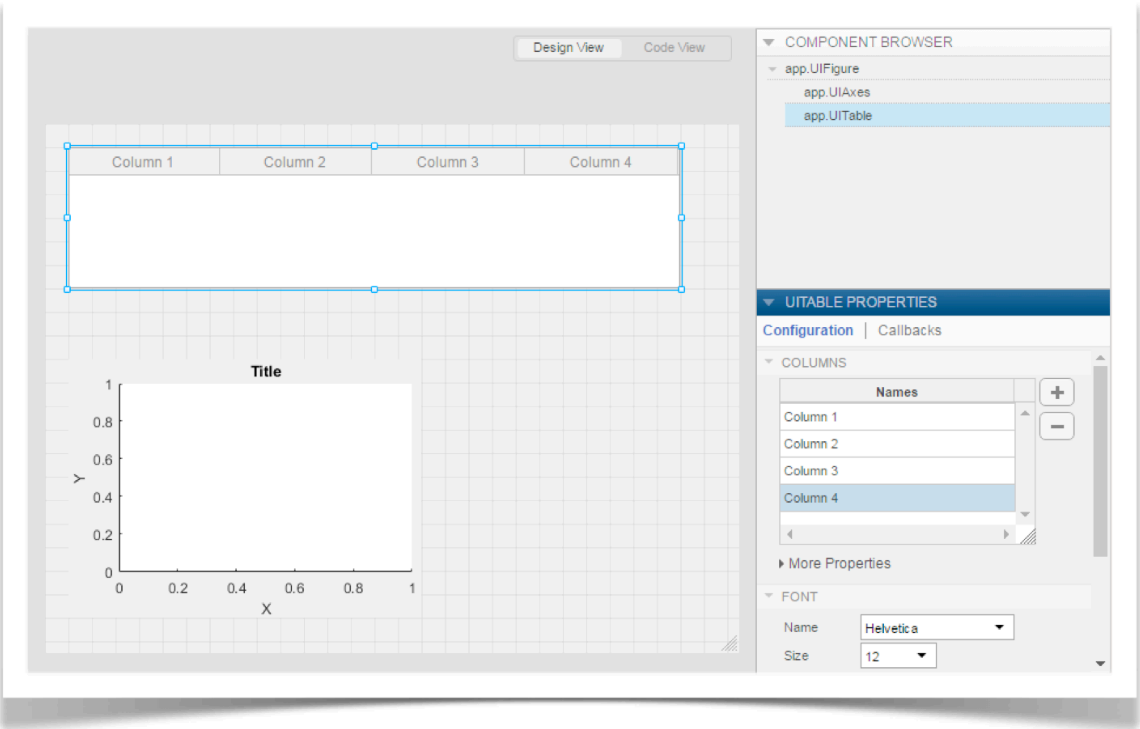


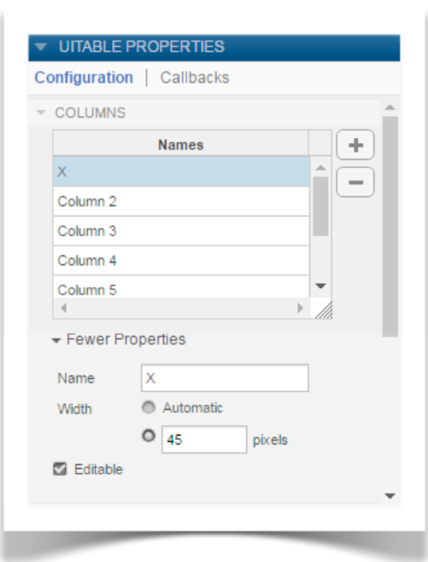



Design View
Code View

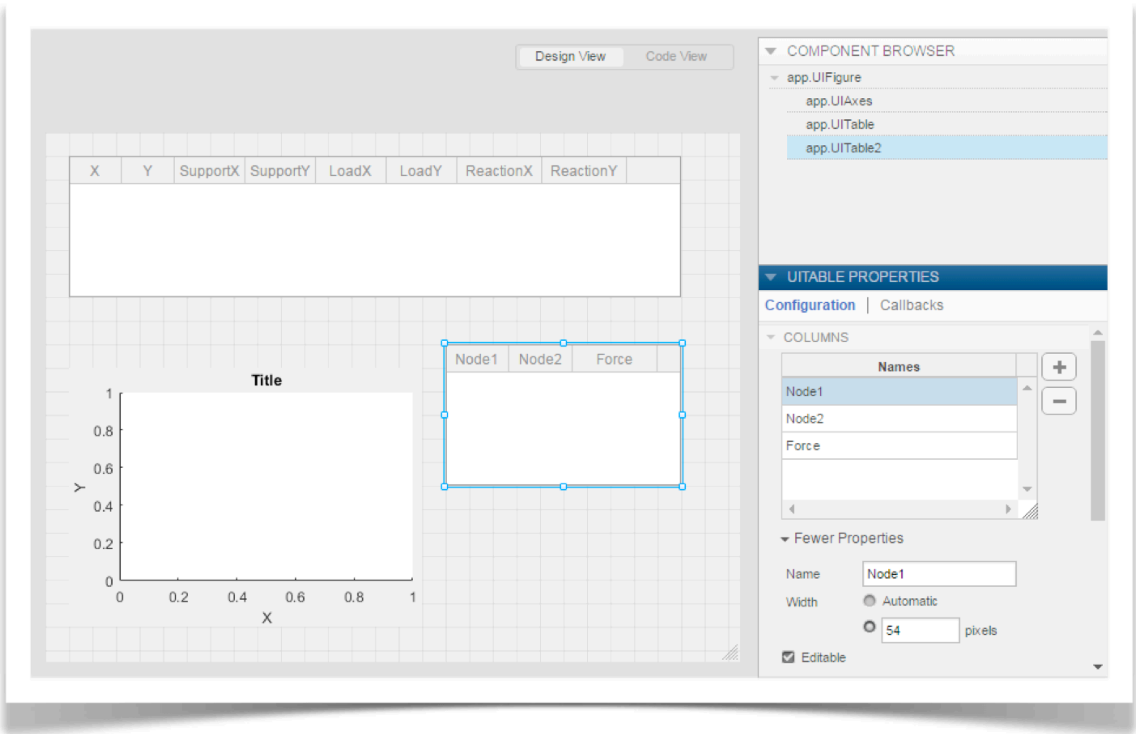

COMPONENT BROWSER
app.UIFigure
UI FIGURE PROPERTIES
Inspector
Callbacks
APPEARANCE
Name
UI Figure
Color
[0.94,0.94,0.94]
PLOTTING
NextPlot
add
Colormap
POSITION
Position
40,20,590,450


DESIGNER
CANVAS
Save
Same Size
Grouping
Evenly
Apply Horizontally
Apply Vertically
Show grid
Snap to grid
Interval
20
Show alignment hints
Show resizing hints
Run
FILE
ALIGN
ARRANGE
SPACE
VIEW
RUN
app1.mlapp*
COMPONENT LIBRARY
COMMON
Axes (2D)
Button
Check Box
Drop Down
Edit Field (Numeric)
Edit Field (Text)
Label
List Box
Radio Button Group
Slider
Spinner
State Button
Table
Design View
Code View
Title
Read more about Graphics Support.
X
Y
COMPONENT BROWSER
app.UIFigure
app.UIAxes
AXES PROPERTIES
Configuration
Callbacks
LABELS
Title
Title
X Label
X
Y Label
Y
APPEARANCE
Grid and Box

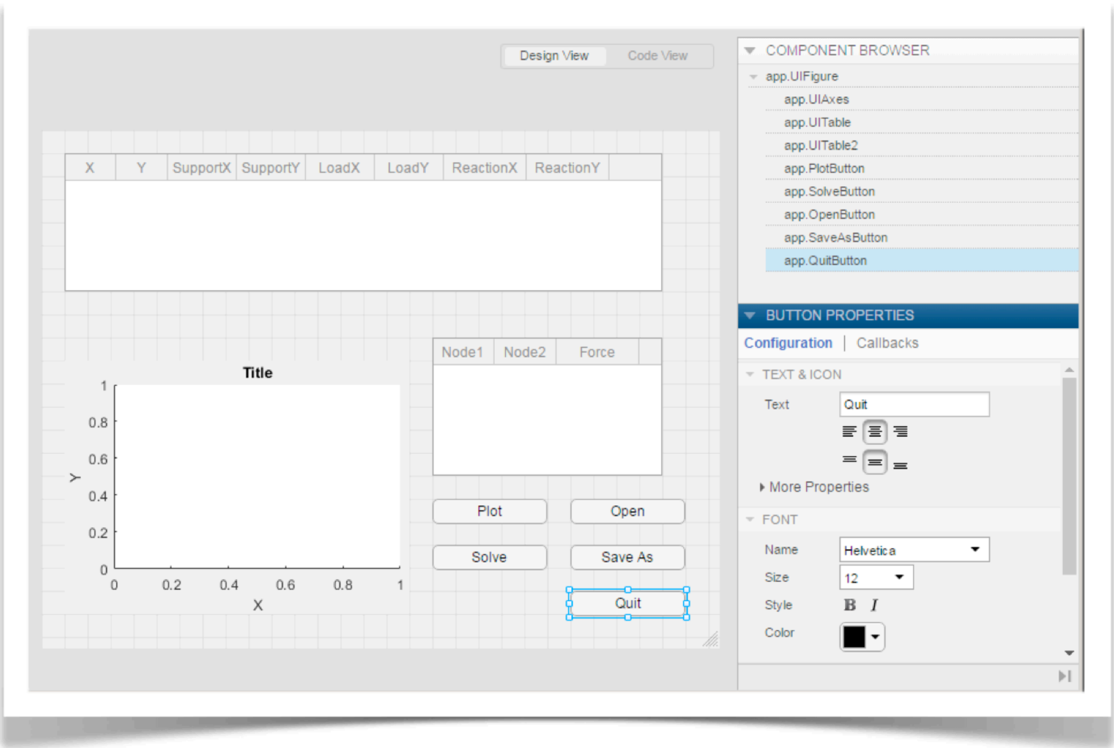


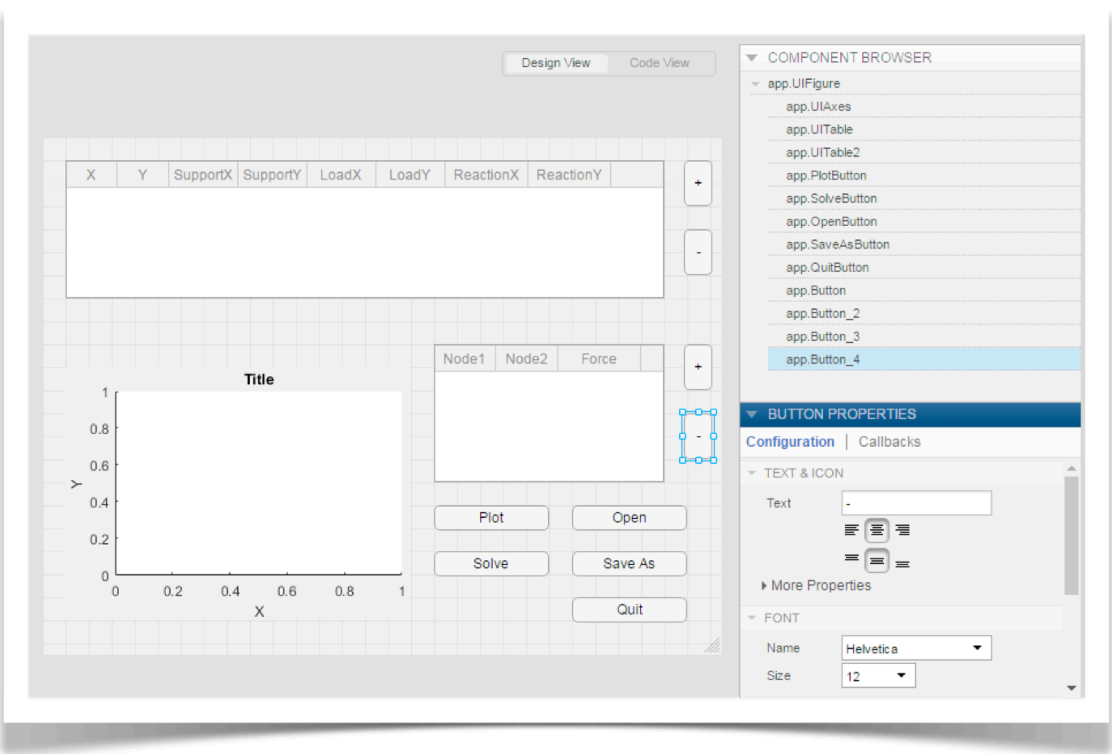


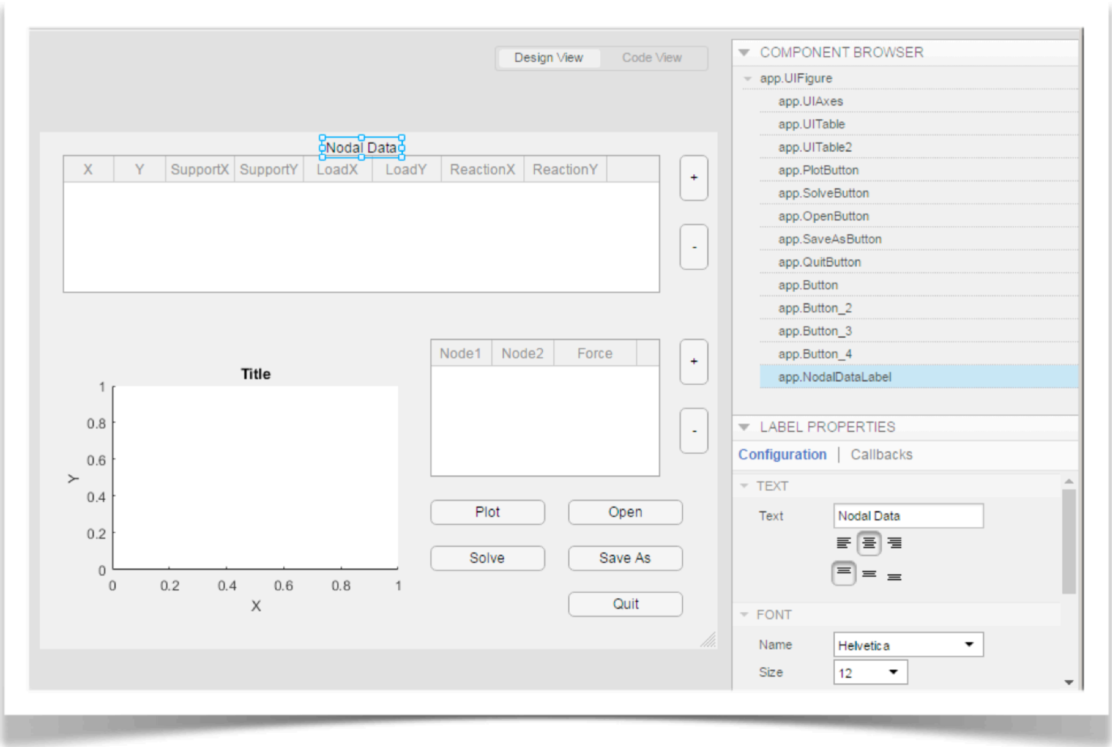
Column	Name	Width	Editable
Column 1	X	45	Yes
Column 2	Y	45	Yes
Column 3	SupportX	60	Yes
Column 4	SupportY	60	Yes
Column 5	LoadX	60	Yes
Column 6	LoadY	60	Yes
Column 7	ReactionX	72	No
Column 8	ReactionY	72	No

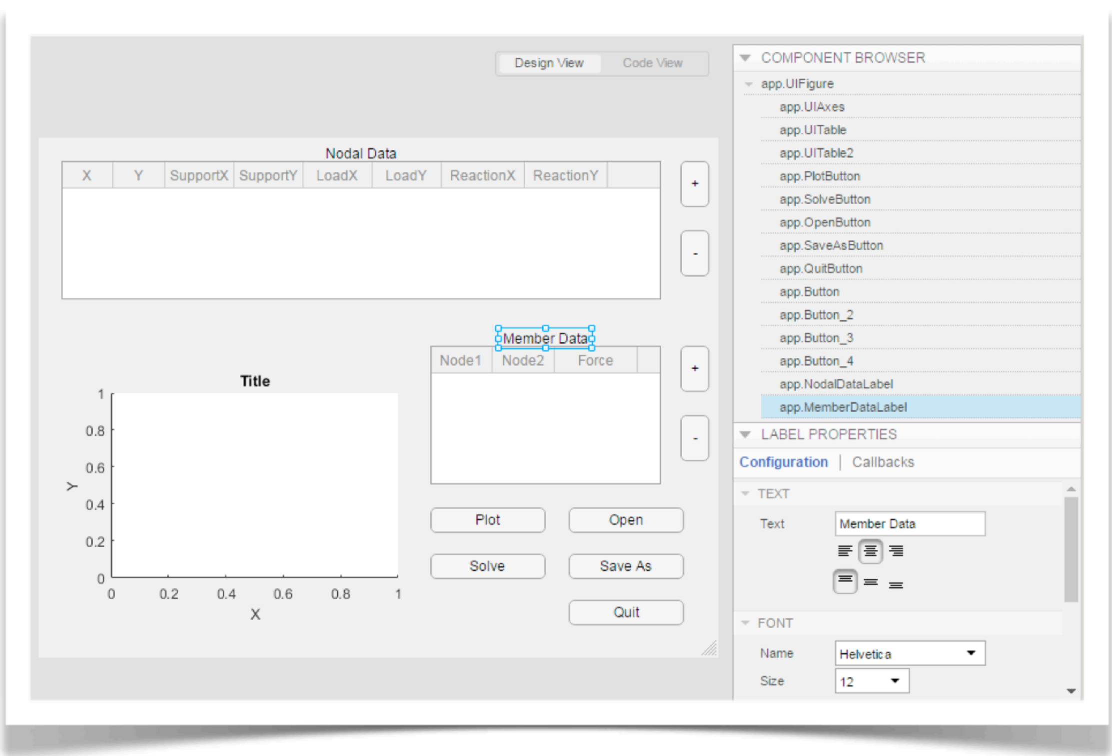


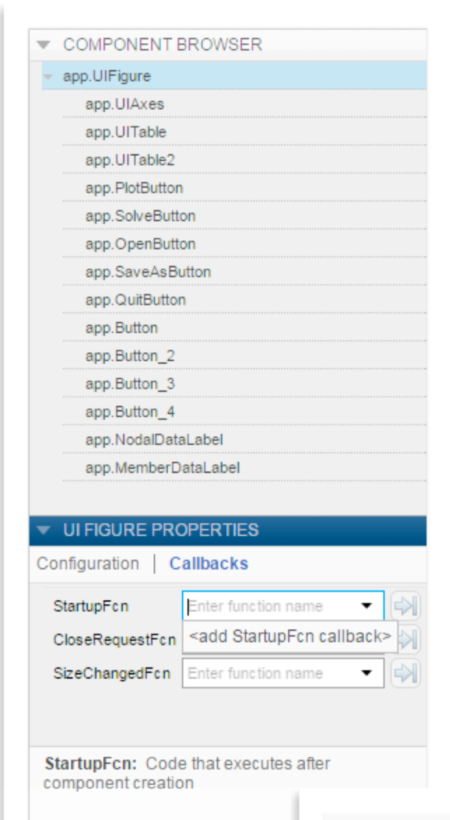
Column	Name	Width	Editable
Column 1	Node1	54	Yes
Column 2	Node2	54	Yes
Column 3	Force	72	No


Design View
Code View
X
Y
SupportX
SupportY
LoadX
LoadY
ReactionX
ReactionY
Node1
Node2
Force
Title
1
0.8
0.6
0.4
0.2
0
Y
0
0.2
0.4
0.6
0.8
1
X
Plot
Open
Solve
Save As
Quit
COMPONENT BROWSER
app.UIFigure
app.UIAxes
app.UITable
app.UITable2
app.PlotButton
app.SolveButton
app.OpenButton
app.SaveAsButton
app.QuitButton
BUTTON PROPERTIES
Configuration
Callbacks
TEXT & ICON
Text
Quit
More Properties
FONT
Name
Helvetica
Size
12
Style
B I
Color

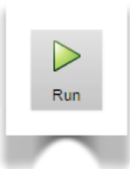

Design View
Code View
X
Y
SupportX
SupportY
LoadX
LoadY
ReactionX
ReactionY
+
-
Node1
Node2
Force
+
-
Title
1
0.8
0.6
0.4
0.2
0
Y
0
0.2
0.4
0.6
0.8
1
X
Plot
Open
Solve
Save As
Quit
COMPONENT BROWSER
app.UIFigure
app.UIAxes
app.UITable
app.UITable2
app.PlotButton
app.SolveButton
app.OpenButton
app.SaveAsButton
app.QuitButton
app.Button
app.Button_2
app.Button_3
app.Button_4
BUTTON PROPERTIES
Configuration
Callbacks
TEXT & ICON
Text
-
More Properties
FONT
Name
Helvetica
Size
12

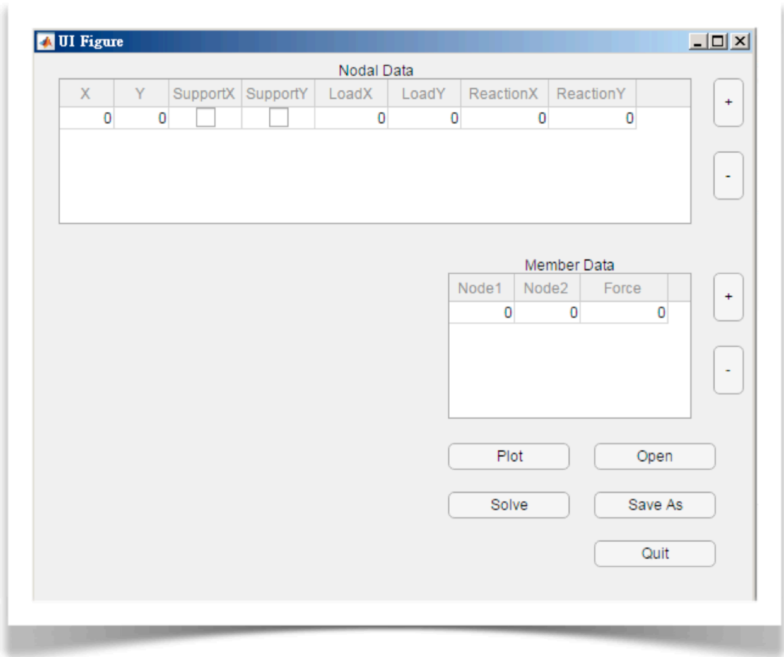

Design View
Code View
Nodal Data
X
Y
SupportX
SupportY
LoadX
LoadY
ReactionX
ReactionY
Node1
Node2
Force
Title
Plot
Open
Solve
Save As
Quit
COMPONENT BROWSER
app.UIFigure
app.UIAxes
app.UITable
app.UITable2
app.PlotButton
app.SolveButton
app.OpenButton
app.SaveAsButton
app.QuitButton
app.Button
app.Button_2
app.Button_3
app.Button_4
app.NodalDataLabel
LABEL PROPERTIES
Configuration | Callbacks
TEXT
Text
Nodal Data
FONT
Name
Helvetica
Size
12

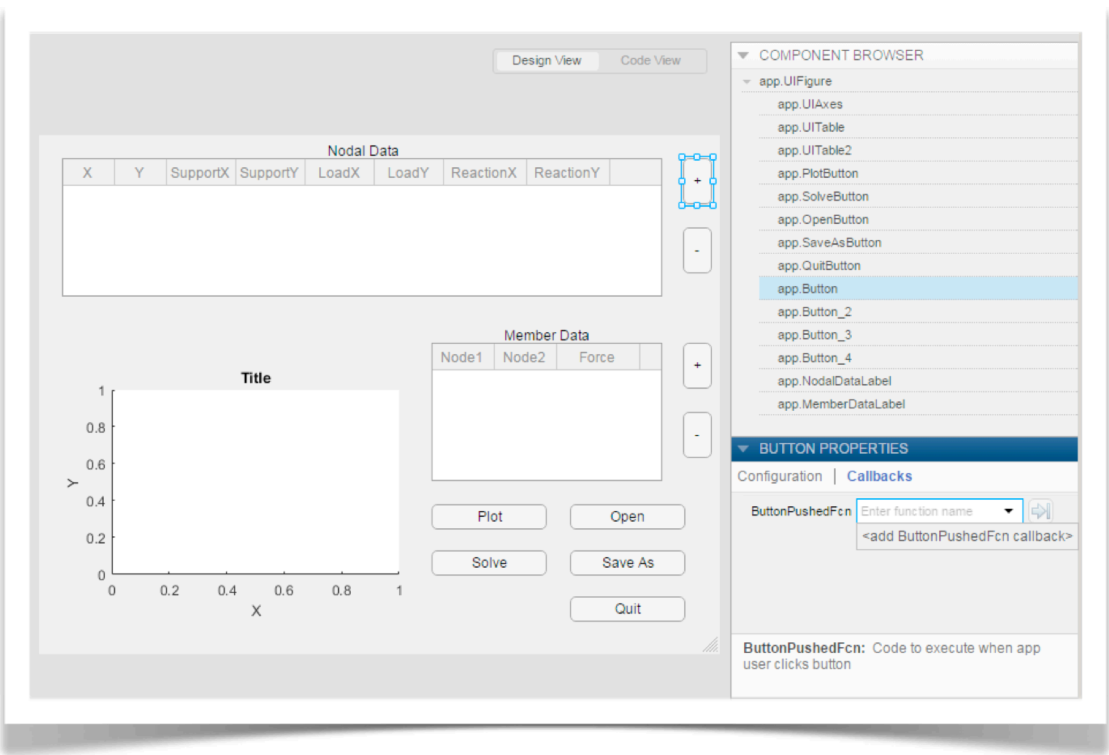

Design View
Code View
Nodal Data
X
Y
SupportX
SupportY
LoadX
LoadY
ReactionX
ReactionY
Member Data
Node1
Node2
Force
Title
Plot
Open
Solve
Save As
Quit
COMPONENT BROWSER
app.UIFigure
app.UIAxes
app.UITable
app.UITable2
app.PlotButton
app.SolveButton
app.OpenButton
app.SaveAsButton
app.QuitButton
app.Button
app.Button_2
app.Button_3
app.Button_4
app.NodalDataLabel
app.MemberDataLabel
LABEL PROPERTIES
Configuration | Callbacks
TEXT
Text
Member Data
FONT
Name
Helvetica
Size
12



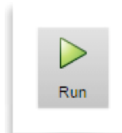

```
        % Code that executes after component creation
        function startupFcn(app)
            global Nodes Members
            Nodes = struct('x', 0, 'y', 0, ...
                'supportx', false, 'supporty', false, ...
                'loadx', 0, 'loady', 0, ...
                'reactionx', 0, 'reactiony', 0);
            Members = struct('node1', 0, 'node2', 0, 'force', 0);
            app.UIAxes.Visible = 'off';
            app.UITable.Data = {0 0 false false 0 0 0 0};
            app.UITable2.Data = {0 0 0};
        end
    end
```

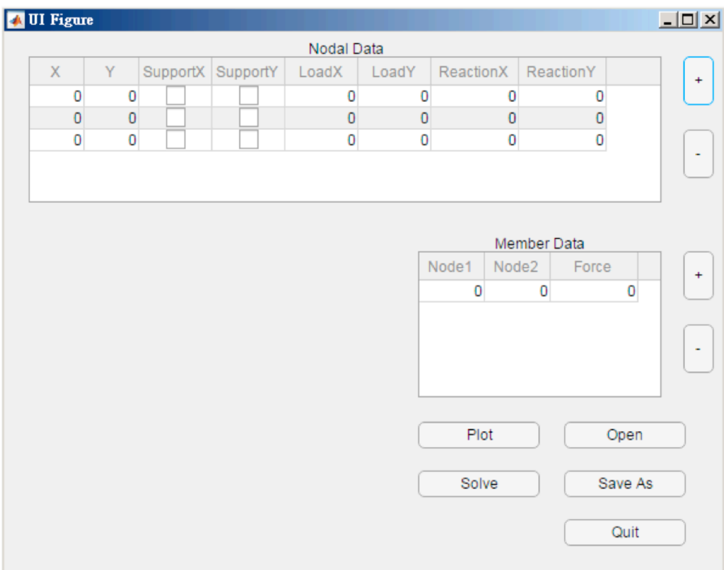

Run


UI Figure
Nodal Data
X
Y
SupportX
SupportY
LoadX
LoadY
ReactionX
ReactionY
+
-
Member Data
Node1
Node2
Force
+
-
Plot
Open
Solve
Save As
Quit

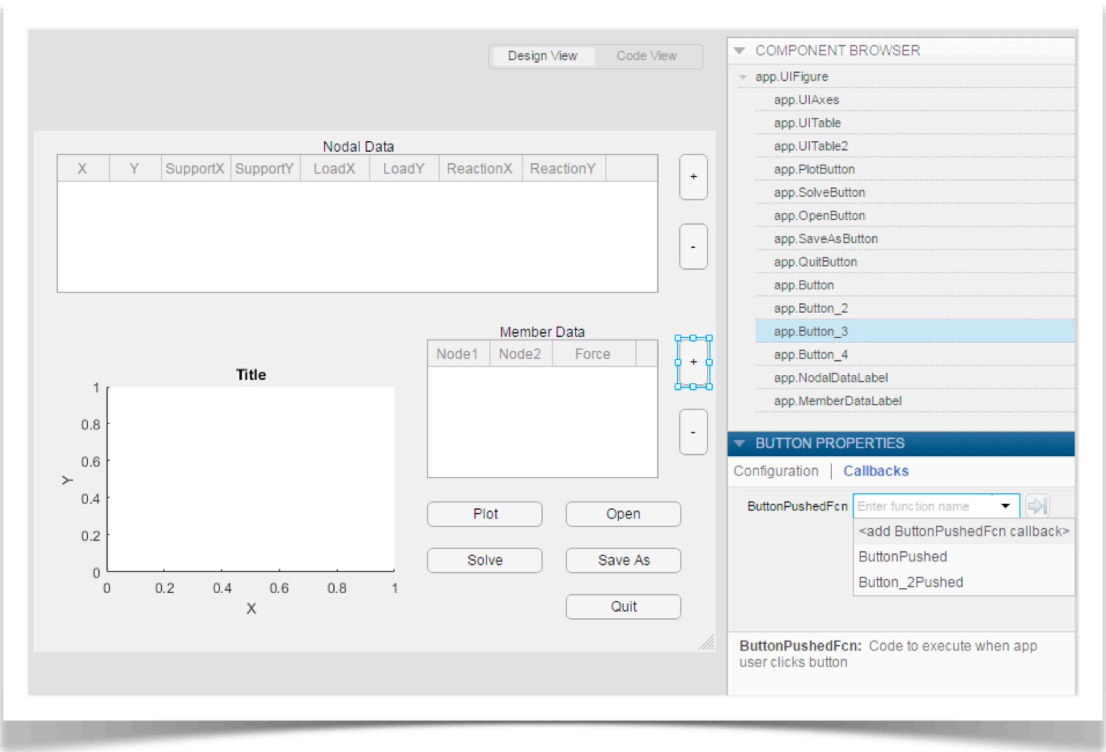

Design View
Code View
Nodal Data
X
Y
SupportX
SupportY
LoadX
LoadY
ReactionX
ReactionY
+
-
Member Data
Node1
Node2
Force
+
-
Title
Y
X
Plot
Open
Solve
Save As
Quit
COMPONENT BROWSER
app.UIFigure
app.UIAxes
app.UITable
app.UITable2
app.PlotButton
app.SolveButton
app.OpenButton
app.SaveAsButton
app.QuitButton
app.Button
app.Button_2
app.Button_3
app.Button_4
app.NodalDataLabel
app.MemberDataLabel
BUTTON PROPERTIES
Configuration
Callbacks
ButtonPushedFcn
Enter function name
<add ButtonPushedFcn callback>
ButtonPushedFcn: Code to execute when app user clicks button

```
        % Button pushed function: Button
        function ButtonPushed(app, event)
            n = size(app.UITable.Data, 1);
            app.UITable.Data(n+1,:) = {0 0 false false 0 0 0 0};
        end
    end
```



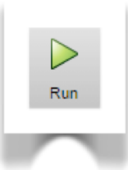


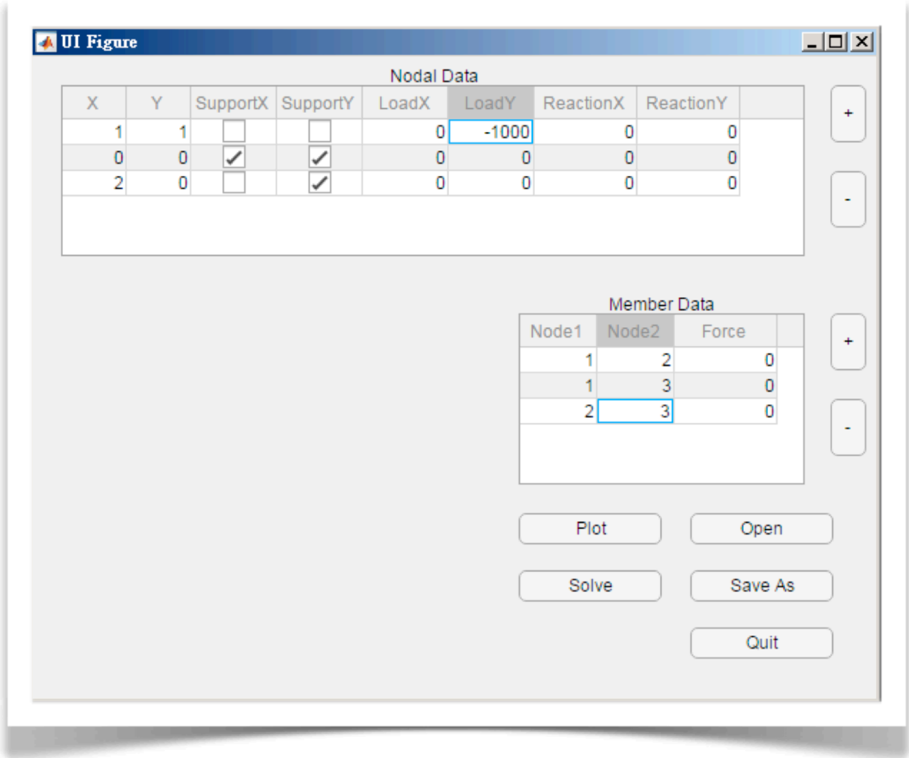

```
        % Button pushed function: Button_2
        function Button_2Pushed(app, event)
            n = size(app.UITable.Data, 1);
            if n > 1
                app.UITable.Data(n,:) = [];
            end
        end
    end
```

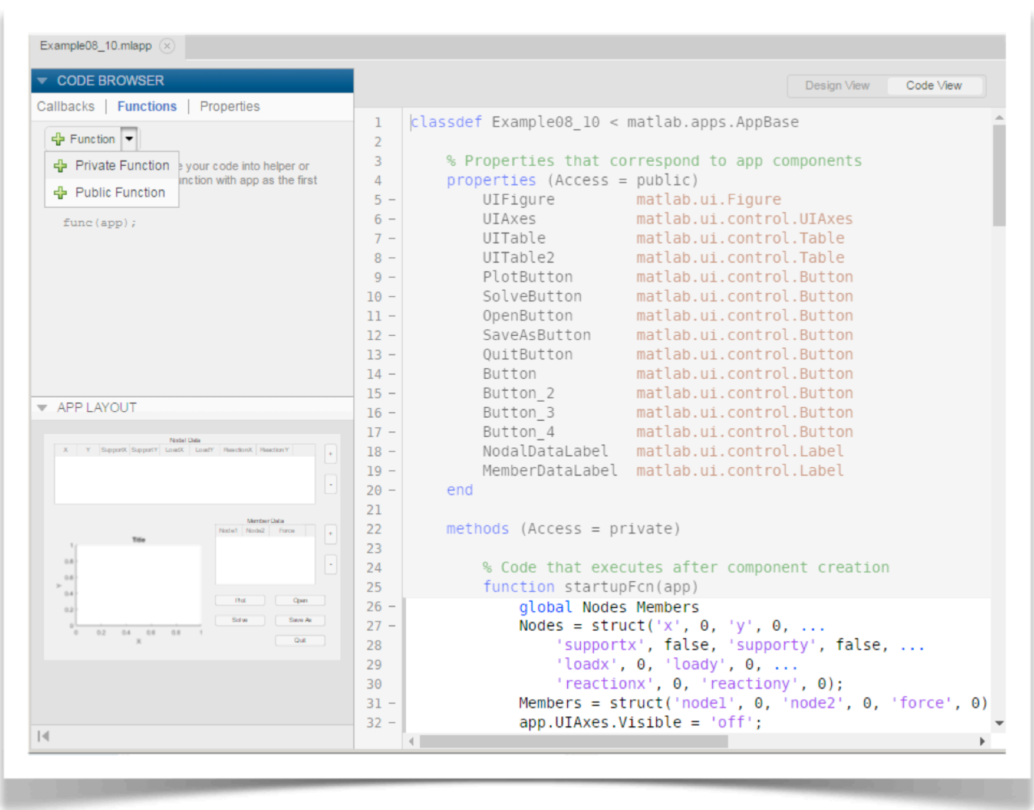



```
        % Button pushed function: Button_3
        function Button_3Pushed(app, event)
            m = size(app.UITable2.Data, 1);
            app.UITable2.Data(m+1,:) = {0 0 0};
        end
    end
```

```
        % Button pushed function: Button_4
        function Button_4Pushed(app, event)
            m = size(app.UITable2.Data, 1);
            if m > 1
                app.UITable2.Data(m,:) = [];
            end
        end
    end
```


Run

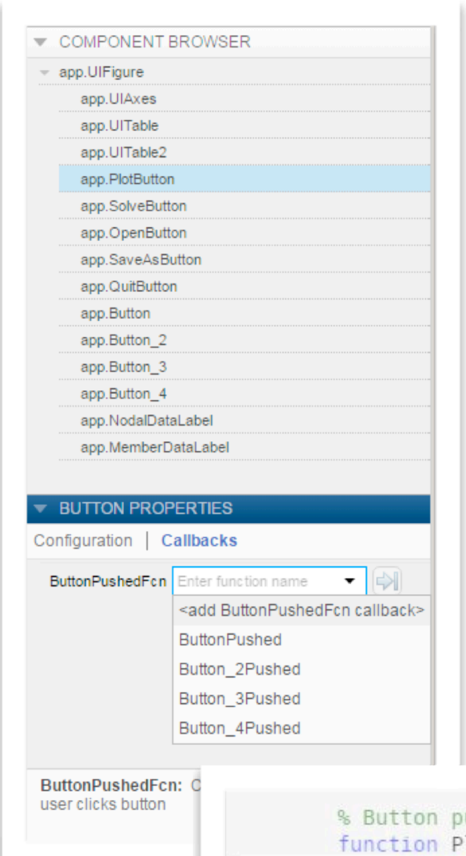

UI Figure
Nodal Data
X
Y
SupportX
SupportY
LoadX
LoadY
ReactionX
ReactionY
1 1 0 -1000 0 0
0 0 0 0 0 0
2 0 0 0 0 0
+
-
Member Data
Node1
Node2
Force
1 2 0
1 3 0
2 3 0
+
-
Plot
Open
Solve
Save As
Quit


Example08_10.mlapp
CODE BROWSER
Callbacks | Functions | Properties
Function
Private Function
Public Function
func(app);
APP LAYOUT
Design View
Code View
classdef Example08_10 < matlab.apps.AppBase
% Properties that correspond to app components
properties (Access = public)
UIFigure matlab.ui.Figure
UIAxes matlab.ui.control.UIAxes
UITable matlab.ui.control.Table
UITable2 matlab.ui.control.Table
PlotButton matlab.ui.control.Button
SolveButton matlab.ui.control.Button
OpenButton matlab.ui.control.Button
SaveAsButton matlab.ui.control.Button
QuitButton matlab.ui.control.Button
Button matlab.ui.control.Button
Button_2 matlab.ui.control.Button
Button_3 matlab.ui.control.Button
Button_4 matlab.ui.control.Button
NodalDataLabel matlab.ui.control.Label
MemberDataLabel matlab.ui.control.Label
end
methods (Access = private)
% Code that executes after component creation
function startupFcn(app)
global Nodes Members
Nodes = struct('x', 0, 'y', 0, ...
'supportx', false, 'supporty', false, ...
'loadx', 0, 'loady', 0, ...
'reactionx', 0, 'reactiony', 0);
Members = struct('node1', 0, 'node2', 0, 'force', 0)
app.UIAxes.Visible = 'off';

```
methods (Access = private)

    function results = func(app)

    end

end
```

```
function plotTruss(app, Nodes, Members)
if (size(fieldnames(Nodes),1)<6 || size(fieldnames(Members),1)<2)
    disp('Truss data not complete'); return
end
n = length(Nodes); m = length(Members);
minX = Nodes(1).x; maxX = Nodes(1).x;
minY = Nodes(1).y; maxY = Nodes(1).y;
for k = 2:n
    if (Nodes(k).x < minX) minX = Nodes(k).x; end
    if (Nodes(k).x > maxX) maxX = Nodes(k).x; end
    if (Nodes(k).y < minY) minY = Nodes(k).y; end
    if (Nodes(k).y > maxY) maxY = Nodes(k).y; end
end
rangeX = maxX-minX; rangeY = maxY-minY;
axis(app.UIAxes, [minX-rangeX/5, maxX+rangeX/5, minY-rangeY/5, maxY+rangeY/5])
delete(app.UIAxes.Children)
axis(app.UIAxes, 'equal', 'off'), title(app.UIAxes, ' ')
hold(app.UIAxes, 'on')
for k = 1:m
    n1 = Members(k).node1; n2 = Members(k).node2;
    x = [Nodes(n1).x, Nodes(n2).x];
    y = [Nodes(n1).y, Nodes(n2).y];
    plot(app.UIAxes, x,y,'k-o', 'MarkerFaceColor', 'k')
end
for k = 1:n
    if Nodes(k).supportx
        x = [Nodes(k).x, Nodes(k).x-rangeX/20, Nodes(k).x-rangeX/20, Nodes(k).x];
        y = [Nodes(k).y, Nodes(k).y+rangeX/40, Nodes(k).y-rangeX/40, Nodes(k).y];
        plot(app.UIAxes, x,y,'k-')
    end
    if Nodes(k).supporty
        x = [Nodes(k).x, Nodes(k).x-rangeX/40, Nodes(k).x+rangeX/40, Nodes(k).x];
        y = [Nodes(k).y, Nodes(k).y-rangeX/20, Nodes(k).y-rangeX/20, Nodes(k).y];
        plot(app.UIAxes, x,y,'k-')
    end
end
end
```

```
function [outNodes, outMembers] = solveTruss(app, Nodes, Members)
n = size(Nodes,2); m = size(Members,2);
if (m+3) < 2*n
    disp('Unstable!')
    outNodes = 0; outMembers = 0; return
elseif (m+3) > 2*n
    disp('Statically indeterminate!')
    outNodes = 0; outMembers = 0; return
end
A = zeros(2*n, 2*n); loads = zeros(2*n,1); nsupport = 0;
for i = 1:n
    for j = 1:m
        if Members(j).node1 == i || Members(j).node2 == i
            if Members(j).node1 == i
                n1 = i; n2 = Members(j).node2;
            elseif Members(j).node2 == i
                n1 = i; n2 = Members(j).node1;
            end
            x1 = Nodes(n1).x; y1 = Nodes(n1).y;
            x2 = Nodes(n2).x; y2 = Nodes(n2).y;
            L = sqrt((x2-x1)^2+(y2-y1)^2);
            A(2*i-1,j) = (x2-x1)/L;
            A(2*i,  j) = (y2-y1)/L;
        end
    end
    if (Nodes(i).supportx == 1)
        nsupport = nsupport+1;
        A(2*i-1,m+nsupport) = 1;
    end
    if (Nodes(i).supporty == 1)
        nsupport = nsupport+1;
        A(2*i, m+nsupport) = 1;
    end
    loads(2*i-1) = -Nodes(i).loadx;
    loads(2*i)   = -Nodes(i).loady;
end
forces = A\loads;
for j = 1:m
    Members(j).force = forces(j);
end
nsupport = 0;
for i = 1:n
    Nodes(i).reactionx = 0;
    Nodes(i).reactiony = 0;
    if (Nodes(i).supportx == 1)
        nsupport = nsupport+1;
        Nodes(i).reactionx = forces(m+nsupport);
    end
    if (Nodes(i).supporty == 1)
        nsupport = nsupport+1;
        Nodes(i).reactiony = forces(m+nsupport);
    end
end
outNodes = Nodes; outMembers = Members;
disp('Solved successfully.')
end
```



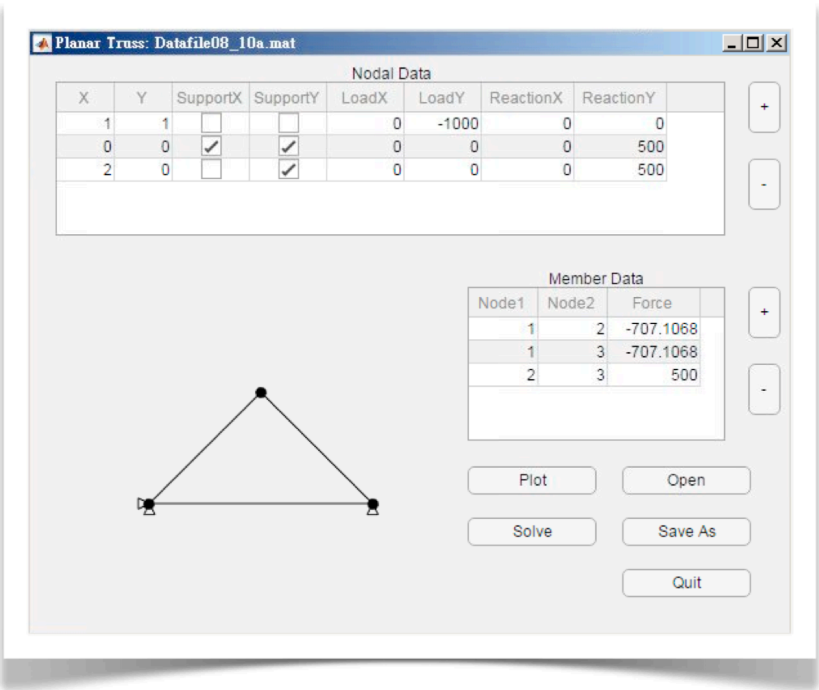

```
        % Button pushed function: PlotButton
        function PlotButtonPushed(app, event)
            global Nodes Members
            Nodes = cell2struct(app.UITable.Data, fieldnames(Nodes), 2)';
            Members = cell2struct(app.UITable2.Data, fieldnames(Members), 2)';
            plotTruss(app, Nodes, Members)
        end
    end
```

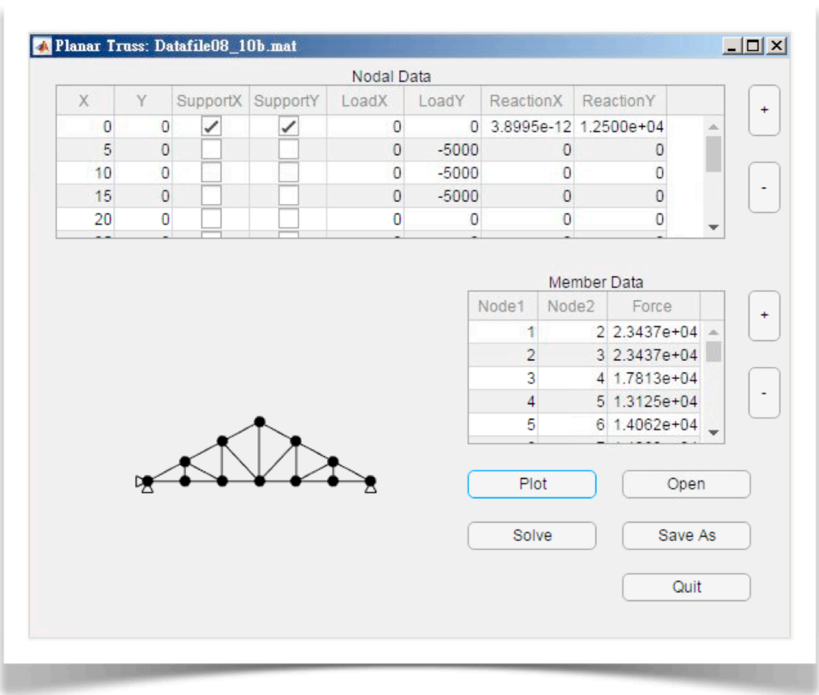
```
        % Button pushed function: SolveButton
        function SolveButtonPushed(app, event)
            global Nodes Members
            Nodes = cell2struct(app.UITable.Data, fieldnames(Nodes), 2)';
            Members = cell2struct(app.UITable2.Data, fieldnames(Members), 2)';
            [Nodes, Members] = solveTruss(app, Nodes, Members);
            app.UITable.Data = permute(struct2cell(Nodes), [1 3 2])';
            app.UITable2.Data = permute(struct2cell(Members), [1 3 2])';
        end
    end
```

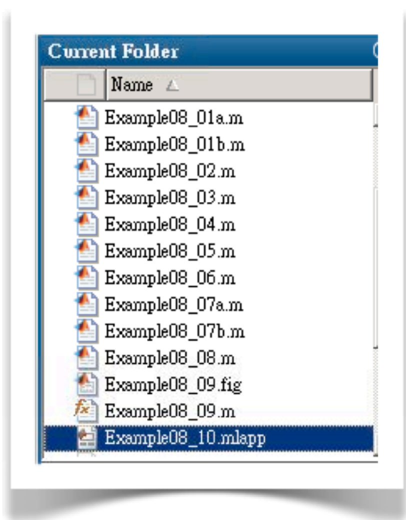
```
        % Button pushed function: OpenButton
        function OpenButtonPushed(app, event)
            global Nodes Members
            [file, path] = uigetfile('*.mat');
            if file
                Nodes = []; Members = [];
                load([path, file])
                app.UITable.Data = permute(struct2cell(Nodes), [1 3 2])';
                app.UITable2.Data = permute(struct2cell(Members), [1 3 2])';
                app.UIFigure.Name = ['Planar Truss: ', file];
            end
        end
    end
```

```
        % Button pushed function: SaveAsButton
        function SaveAsButtonPushed(app, event)
            global Nodes Members
            [file, path] = uiputfile('*.mat');
            if file
                Nodes = cell2struct(app.UITable.Data, fieldnames(Nodes), 2)';
                Members = cell2struct(app.UITable2.Data, fieldnames(Members), 2)';
                save([path, file], 'Nodes', 'Members')
                app.UIFigure.Name = ['Planar Truss: ', file];
            end
        end
```

```
        % Button pushed function: QuitButton
        function QuitButtonPushed(app, event)
            close(app.UIFigure)
        end
    end
```

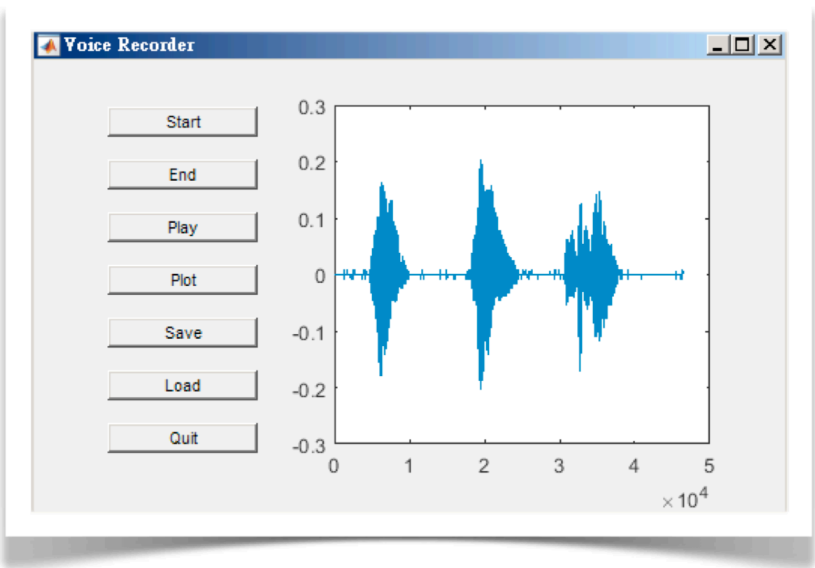

Planar Truss: Datafile08_10a.mat
Nodal Data
X Y SupportX SupportY LoadX LoadY ReactionX ReactionY
1 1 0 -1000 0 0
0 0 0 0 0 500
2 0 0 0 0 500
Member Data
Node1 Node2 Force
1 2 -707.1068
1 3 -707.1068
2 3 500
Plot
Open
Solve
Save As
Quit
+
-


Planar Truss: Datafile08_10b.mat
Nodal Data
X Y SupportX SupportY LoadX LoadY ReactionX ReactionY
0 0 0 0 3.8995e-12 1.2500e+04
5 0 0 -5000 0 0
10 0 0 -5000 0 0
15 0 0 -5000 0 0
20 0 0 0 0 0
Member Data
Node1 Node2 Force
1 2 2.3437e+04
2 3 2.3437e+04
3 4 1.7813e+04
4 5 1.3125e+04
5 6 1.4062e+04
Plot
Open
Solve
Save As
Quit
+
-


Current Folder
Name
Example08_01a.m
Example08_01b.m
Example08_02.m
Example08_03.m
Example08_04.m
Example08_05.m
Example08_06.m
Example08_07a.m
Example08_07b.m
Example08_08.m
Example08_09.fig
Example08_09.m
Example08_10.mlapp

```
classdef Example08_10
    properties (Access = public)
        UIFigure    ...
        ...
    end

    methods (Access = private)
        function solveTruss
        ...

        function plotTruss
        ...
    end

    methods (Access = private)
        function startupFcn
        ...

        function ButtonPushed
        ...

        function Button_2Pushed
        ...

        function Button_3Pushed
        ...

        function Button_4Pushed
        ...

        function OpenButtonPushed
        ...

        function PlotButtonPushed
        ...

        function QuitButtonPushed
        ...

        function SaveAsQuitButtonPushed
        ...

        function SolveButtonPushed
        ...
    end

    methods (Access = private)
        function createComponents(app)
        ...
    end

    methods (Access = public)
        function app = Example08_10
        ...

        function delete
        ...
    end
end
```

8.11 Additional Exercise Problems



Chapter 9

Symbolic Mathematics

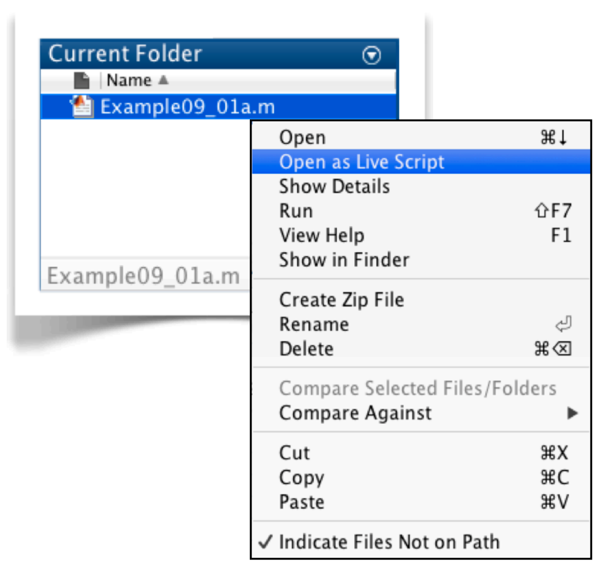
MATLAB can process not only numeric data but also symbolic expressions. To use symbolic mathematics, you need a license that includes Symbolic Math Toolbox. This chapter assumes that you have a license that includes Symbolic Math Toolbox.

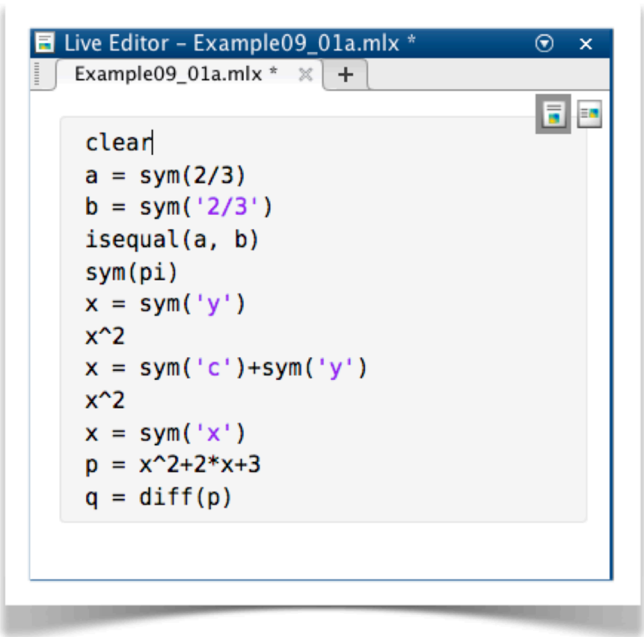
9.1 Symbolic Numbers, Variables, Functions, and Expressions

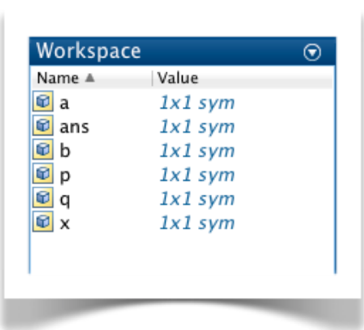
Example09_01a.m: Basic Concepts

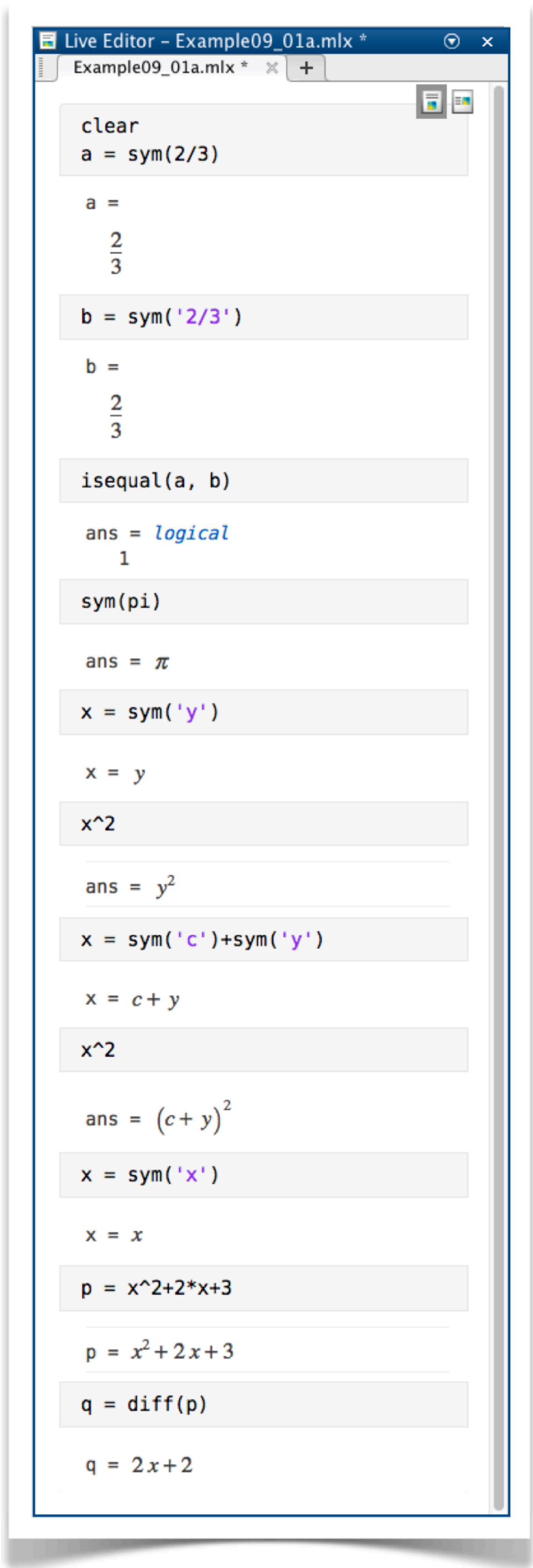
[2] Type the following commands and save as Example09_01a.m. These commands demonstrate the creation of symbolic numbers, symbolic variables, and symbolic expressions, using the functions `sym`.

```
clear
a = sym(2/3)
b = sym('2/3')
isequal(a, b)
sym(pi)
x = sym('y')
x^2
x = sym('c')+sym('y')
x^2
x = sym('x')
p = x^2+2*x+3
q = diff(p)
```




Live Editor – Example09_01a.mlx *
Example09_01a.mlx *
clear
a = sym(2/3)
b = sym('2/3')
isequal(a, b)
sym(pi)
x = sym('y')
x^2
x = sym('c')+sym('y')
x^2
x = sym('x')
p = x^2+2*x+3
q = diff(p)


Workspace
Name ▲
Value
a
1x1 sym
ans
1x1 sym
b
1x1 sym
p
1x1 sym
q
1x1 sym
x
1x1 sym


Live Editor – Example09_01a.mlx *
Example09_01a.mlx *
clear
a = sym(2/3)
a =
2/3
b = sym('2/3')
b =
2/3
isequal(a, b)
ans = logical
1
sym(pi)
ans = π
x = sym('y')
x = y
x^2
ans = y^2
x = sym('c')+sym('y')
x = c + y
x^2
ans = (c + y)^2
x = sym('x')
x = x
p = x^2+2*x+3
p = x^2 + 2x + 3
q = diff(p)
q = 2x + 2

Example09_01b.m: Symbolic Expressions

[8] As exercises, let's create the following expressions: $\rightarrow$

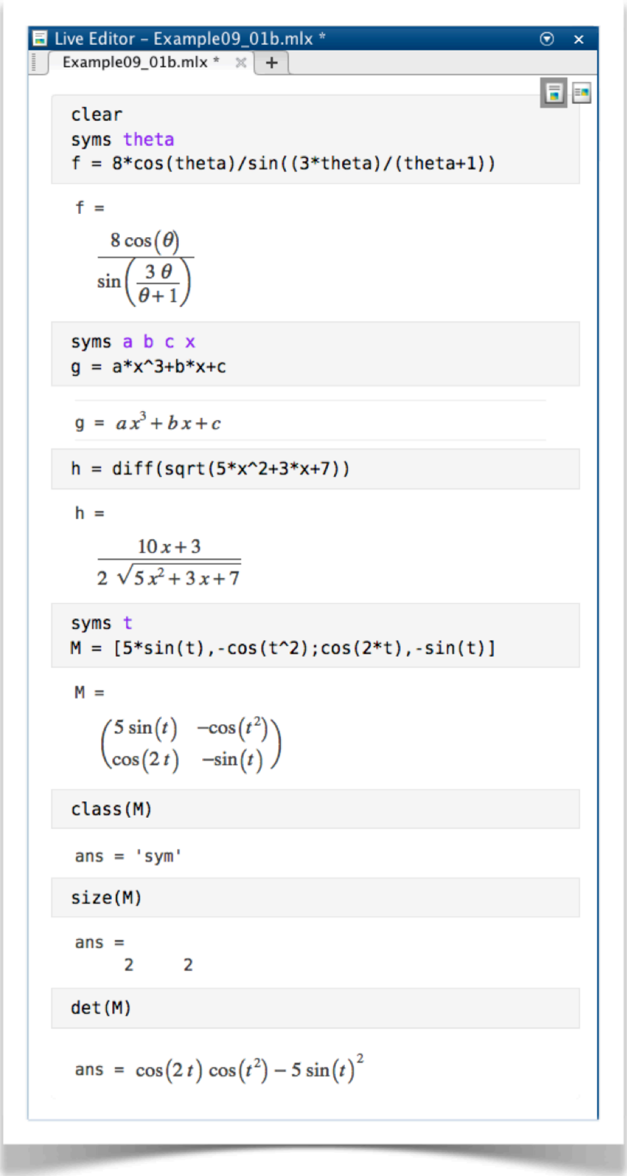
$$f = \frac{8\cos\theta}{\sin\left(\frac{3\theta}{\theta+1}\right)} \quad \text{(a)}$$

$$g = ax^3 + bx + c \quad \text{(b)}$$

$$h = \frac{d}{dx}\sqrt{5x^2+3x+7} = \frac{10x+3}{2\sqrt{5x^2+3x+7}} \quad \text{(c)}$$

$$M = \begin{bmatrix} 5\sin t & -\cos t^2 \\ \cos 2t & -\sin t \end{bmatrix} \quad \text{(d)}$$

```
clear
syms theta
f = 8*cos(theta)/sin((3*theta)/(theta+1))
syms a b c x
g = a*x^3+b*x+c
h = diff(sqrt(5*x^2+3*x+7))
syms t
M = [5*sin(t),-cos(t^2);cos(2*t),-sin(t)]
class(M)
size(M)
det(M)
```


Live Editor – Example09_01b.mlx *

Example09_01b.mlx *

```
clear
syms theta
f = 8*cos(theta)/sin((3*theta)/(theta+1))
```

f =

$$\frac{8\cos(\theta)}{\sin\left(\frac{3\,\theta}{\theta+1}\right)}$$

```
syms a b c x
g = a*x^3+b*x+c
```

g = $a\,x^3 + b\,x + c$

```
h = diff(sqrt(5*x^2+3*x+7))
```

h =

$$\frac{10\,x+3}{2\,\sqrt{5\,x^2+3\,x+7}}$$

```
syms t
M = [5*sin(t),-cos(t^2);cos(2*t),-sin(t)]
```

M =

$$\begin{pmatrix} 5\sin(t) & -\cos(t^2) \\ \cos(2\,t) & -\sin(t) \end{pmatrix}$$

```
class(M)
```

ans = 'sym'

```
size(M)
```

```
ans =
     2     2
```

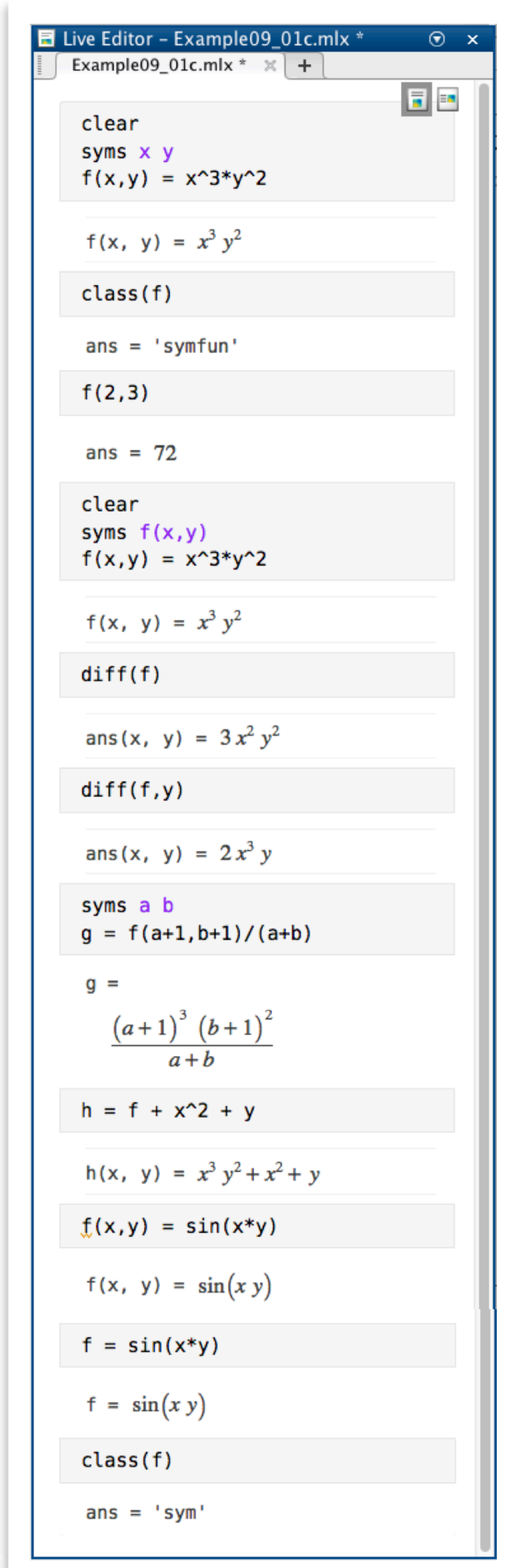
```
det(M)
```

ans = $\cos(2\,t)\cos(t^2) - 5\sin(t)^2$

Example09_01c.m: Symbolic Functions

[11] Following commands demonstrate the creation and the use of symbolic functions.

```
clear
syms x y
f(x,y) = x^3*y^2
class(f)
f(2,3)
clear
syms f(x,y)
f(x,y) = x^3*y^2
diff(f)
diff(f,y)
syms a b
g = f(a+1,b+1)/(a+b)
h = f + x^2 + y
f(x,y) = sin(x*y)
f = sin(x*y)
class(f)
```



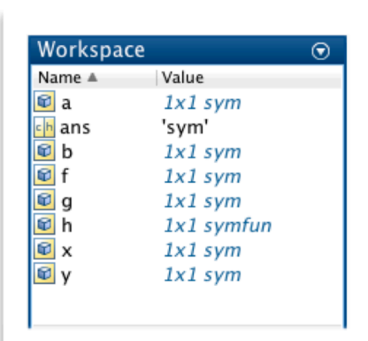


Table 9.1 Creation of Symbolic Constant, Variables, and Functions

Functions	Description
`syms v1 v2 ...`	Create symbolic variables and functions
`s = sym(string)`	Create symbolic constants, variables, and functions.
`s = sym(number)`	Create symbolic constants, variables, and functions.
`v = symvar(s,n)`	Return n variables closest to x

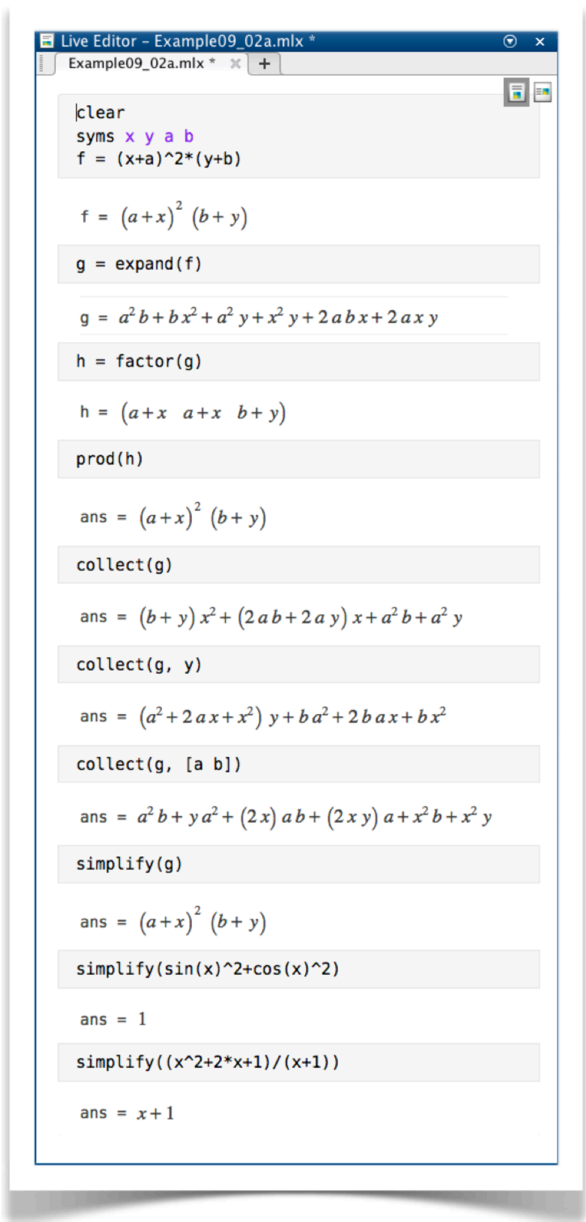
Details and More: Help>Symbolic Math Toolbox>Symbolic Computations in MATLAB>Symbolic Variables, Expressions, Functions, and Preferences>Create Symbolic Variables, Expressions, and Functions

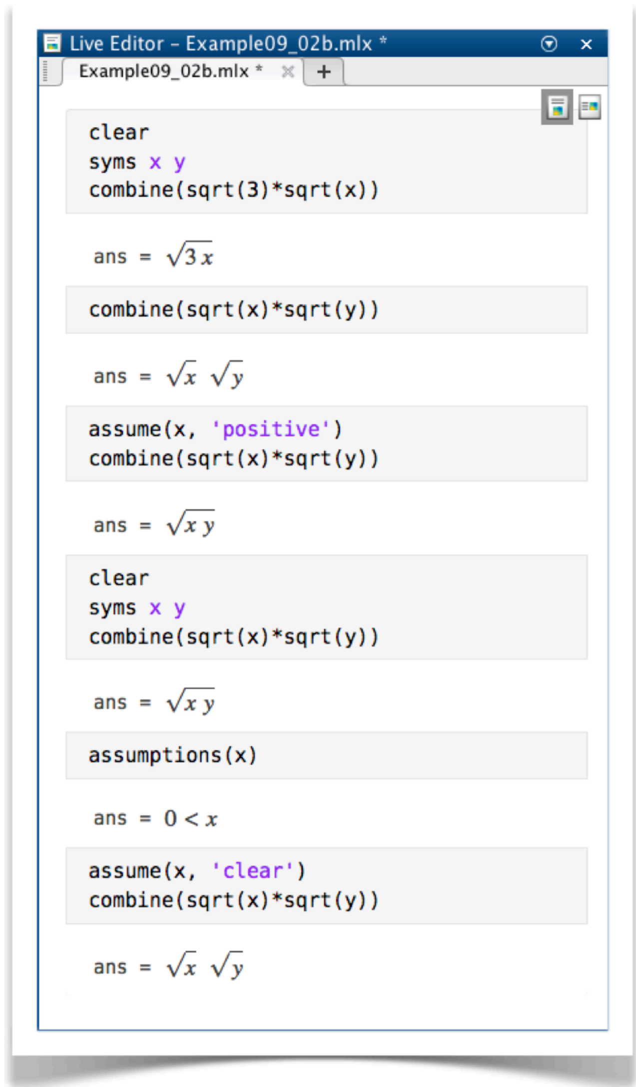
9.2 Simplification of Expressions

Example09_02a.m: Simplification of Expressions

[2] The following commands demonstrate the use of some simplification functions.

```
clear
syms x y a b
f = (x+a)^2*(y+b)
g = expand(f)
h = factor(g)
prod(h)
collect(g)
collect(g, y)
collect(g, [a b])
simplify(g)
simplify(sin(x)^2+cos(x)^2)
simplify((x^2+2*x+1)/(x+1))
```





Example09_02b.m: `combine`

[5] The following commands demonstrate the use of the function `combine` to simplify expressions. They also show the use of the function `assume` to set assumptions on symbolic variables.

```
clear
syms x y
combine(sqrt(3)*sqrt(x))
combine(sqrt(x)*sqrt(y))
assume(x, 'positive')
combine(sqrt(x)*sqrt(y))
clear
syms x y
combine(sqrt(x)*sqrt(y))
assumptions(x)
assume(x, 'clear')
combine(sqrt(x)*sqrt(y))
```

Table 9.2 Simplification of Expressions

Functions	Description
`s = expand(expr,name,value)`	Expand expressions
`f = factor(expr,name,value)`	Factor expressions
`s = collect(expr,var)`	Collect terms with same powers
`s = combine(expr,name,value)`	Combine terms of same algebraic structures
`s = simplify(expr,name,value)`	General simplification
`s = simplifyFraction(expr,name,value)`	Compute normal forms of rational expressions
`assume(expr,set)`	Set assumption on symbolic variables
`assumptions(var)`	Show assumptions of symbolic variables

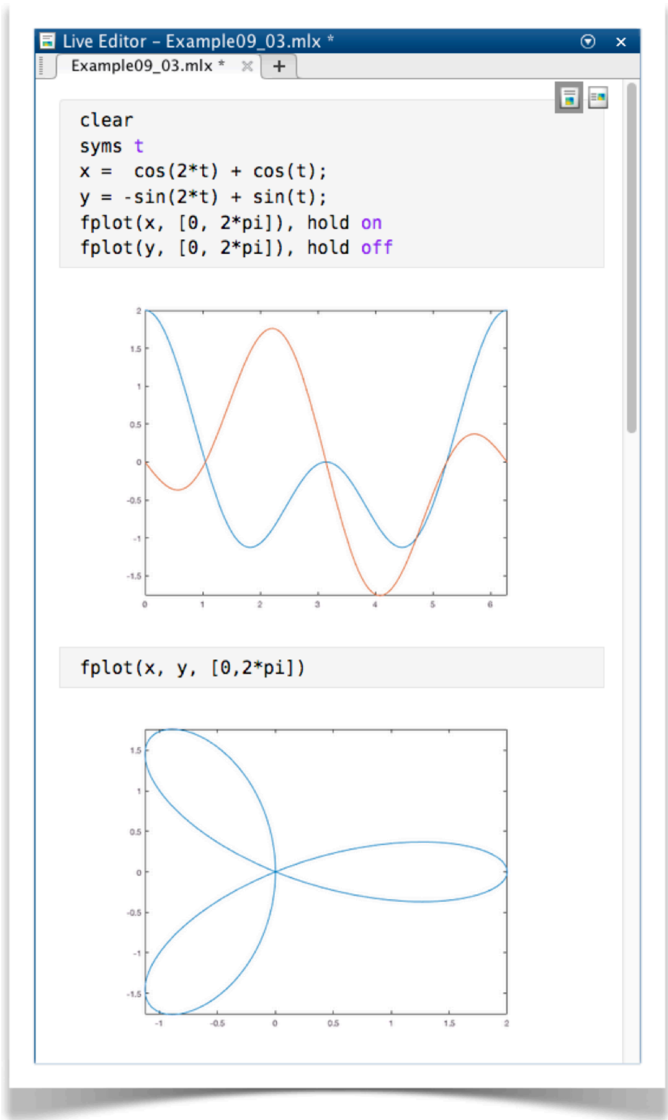
Details and More: Help>Symbolic Math Toolbox>Mathematics>Formula Manipulation and Simplification

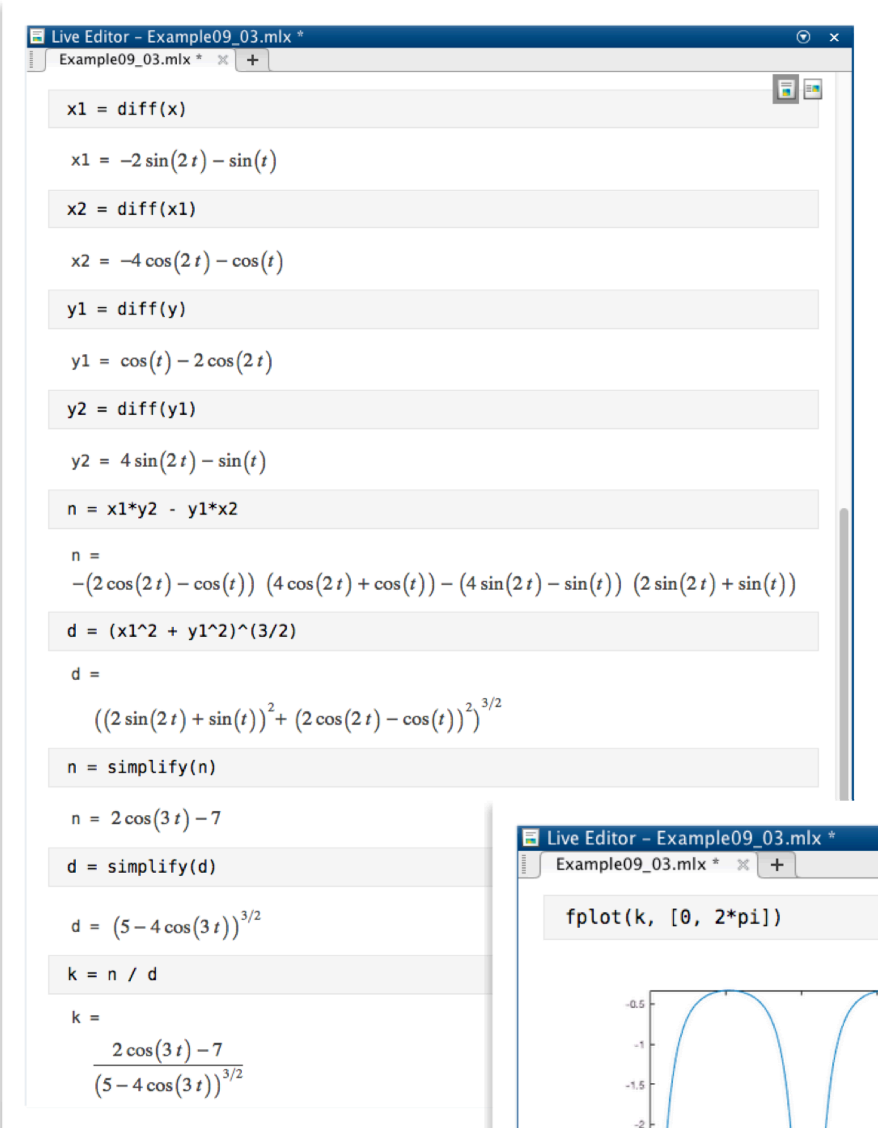
9.3 Symbolic Differentiation: Curvature of a Curve

Example09_03.m: Curvature of a Planar Curve

[2] The following commands calculate the curvature of a curve given in [1].

```
clear
syms t
x =  cos(2*t) + cos(t);
y = -sin(2*t) + sin(t);
fplot(x, [0, 2*pi]), hold on
fplot(y, [0, 2*pi]), hold off
fplot(x, y, [0,2*pi])
x1 = diff(x)
x2 = diff(x1)
y1 = diff(y)
y2 = diff(y1)
n = x1*y2 - y1*x2
d = (x1^2 + y1^2)^(3/2)
n = simplify(n)
d = simplify(d)
k = n / d
fplot(k, [0, 2*pi])
k = subs(k, cos(3*t), 'C')
```




Live Editor – Example09_03.mlx *
Example09_03.mlx *
x1 = diff(x)
x1 = $-2\sin(2t) - \sin(t)$
x2 = diff(x1)
x2 = $-4\cos(2t) - \cos(t)$
y1 = diff(y)
y1 = $\cos(t) - 2\cos(2t)$
y2 = diff(y1)
y2 = $4\sin(2t) - \sin(t)$
n = x1*y2 - y1*x2
n =
$-(2\cos(2t) - \cos(t))\ (4\cos(2t) + \cos(t)) - (4\sin(2t) - \sin(t))\ (2\sin(2t) + \sin(t))$
d = (x1^2 + y1^2)^(3/2)
d =
$((2\sin(2t) + \sin(t))^2 + (2\cos(2t) - \cos(t))^2)^{3/2}$
n = simplify(n)
n = $2\cos(3t) - 7$
d = simplify(d)
d = $(5 - 4\cos(3t))^{3/2}$
k = n / d
k =
$\frac{2\cos(3t) - 7}{(5 - 4\cos(3t))^{3/2}}$

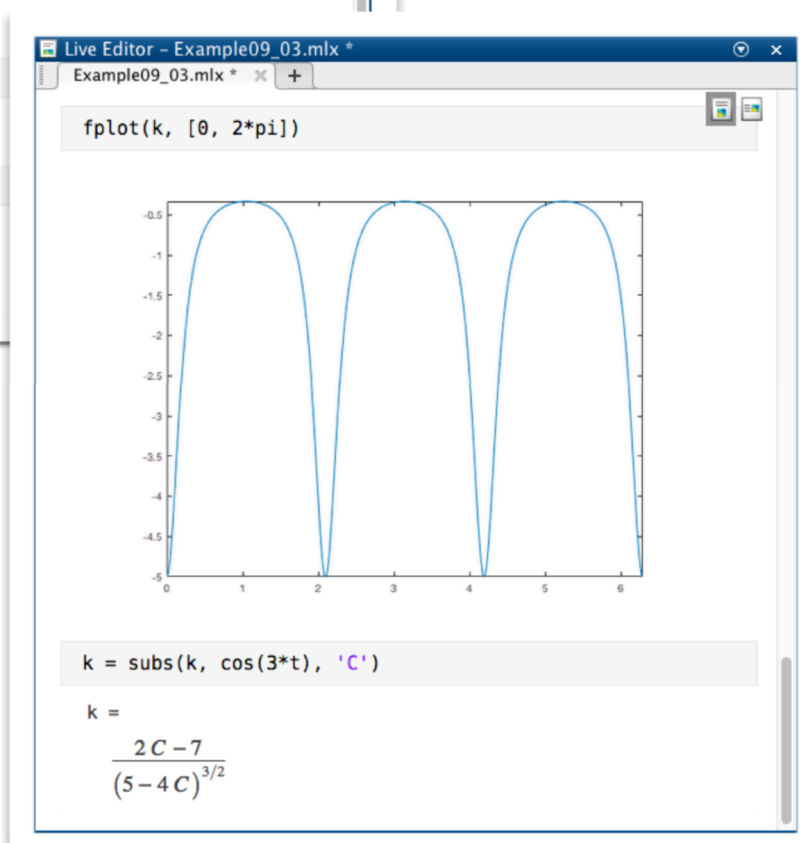

Live Editor – Example09_03.mlx *
Example09_03.mlx *
fplot(k, [0, 2*pi])
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
-4
-4.5
-5
0
1
2
3
4
5
6
k = subs(k, cos(3*t), 'C')
k =
$\frac{2C - 7}{(5 - 4C)^{3/2}}$

Table 9.3 Line Plots

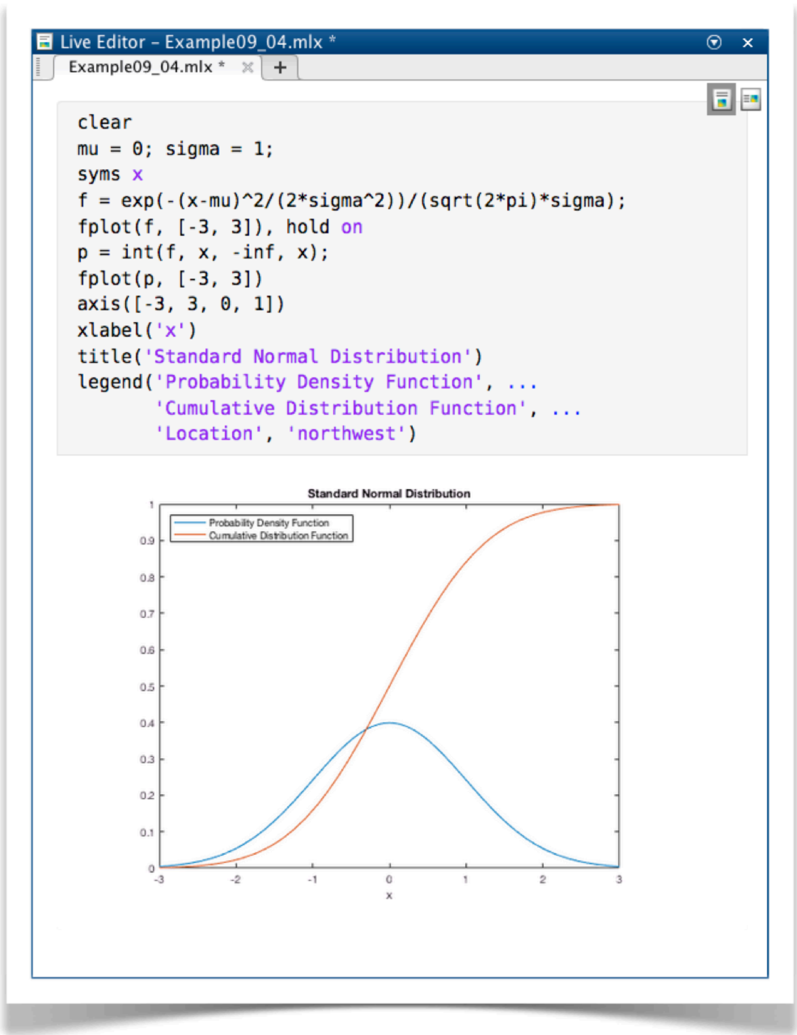
Functions	Description
`fplot(fun,lineSpec)`	Plot expression or function
`fplot3(funx,funy,funz,lineSpec)`	3-D parametric curve plotter
`fimplicit(fun,lineSpec)`	Plot implicit function
`fimplicit3(fun,lineSpec)`	Plot 3-D implicit function
Details and More: Help>MATLAB>Graphics>2-D and 3-D Plots>Line Plots	

9.4 Symbolic Integration: Normal Distributions

Example09_04.m: Normal Distribution Curves

[2] This script plots a p.d.f. curve, Eq. (a), for the standard normal distribution. It also integrates the p.d.f. using the function `int` to obtain a c.d.f., Eq. (b), and plots a c.d.f. curve. →

```
clear
mu = 0; sigma = 1;
syms x
f = exp(-(x-mu)^2/(2*sigma^2))/(sqrt(2*pi)*sigma);
fplot(f, [-3, 3]), hold on
p = int(f, x, -inf, x);
fplot(p, [-3, 3])
axis([-3, 3, 0, 1])
xlabel('x')
title('Standard Normal Distribution')
legend('Probability Density Function', ...
       'Cumulative Distribution Function', ...
       'Location', 'northwest')
```


Live Editor – Example09_04.mlx *
Example09_04.mlx *
clear
mu = 0; sigma = 1;
syms x
f = exp(-(x-mu)^2/(2*sigma^2))/(sqrt(2*pi)*sigma);
fplot(f, [-3, 3]), hold on
p = int(f, x, -inf, x);
fplot(p, [-3, 3])
axis([-3, 3, 0, 1])
xlabel('x')
title('Standard Normal Distribution')
legend('Probability Density Function', ...
'Cumulative Distribution Function', ...
'Location', 'northwest')
Standard Normal Distribution
Probability Density Function
Cumulative Distribution Function
x

9.5 Limits

Example09_05.m: Limits

[2] The following commands demonstrate the use of the function `limit`.

```
clear
syms x h n
f(x) = sin(x);
g(x) = limit((f(x+h)-f(x))/h, h, 0)
g(x) = diff(f)
f(x) = x^n;
g(x) = limit((f(x+h)-f(x))/h, h, 0)
g(x) = diff(f)
limit(sin(x)/x, x, 0)
f(x) = limit((1+x/n)^n, n, inf)
vpa(f(1))
```

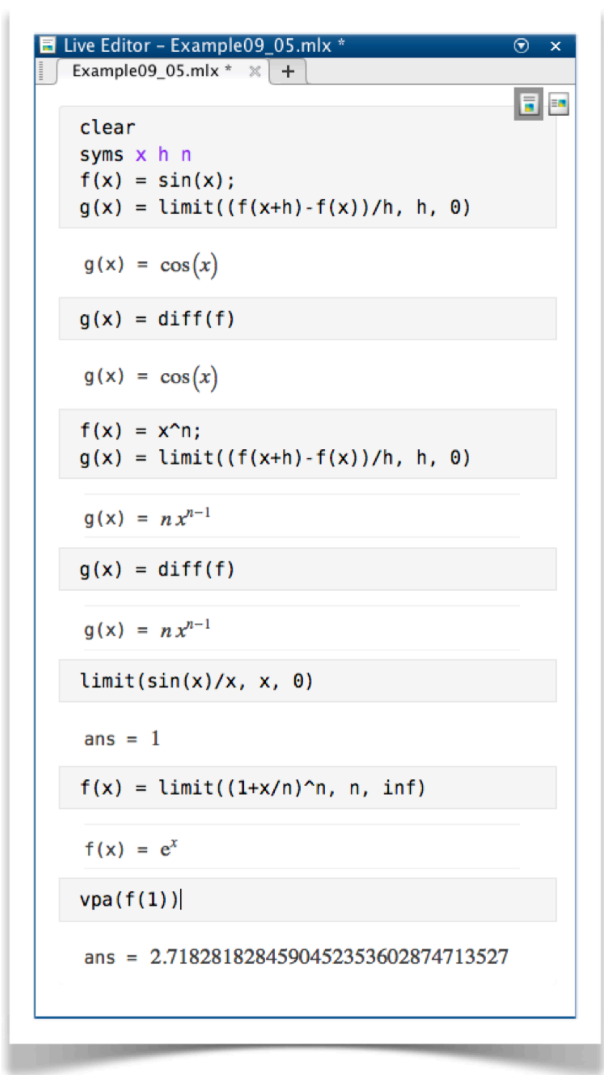


Table 9.5 Limits	
Functions	Description
`limit(expr,x,a)`	Compute limit of symbolic expression
Details and More: Help>Symbolic Math Toolbox>Mathematics>Calculus>Limits	

9.6 Taylor Series

Example09_06.m: Taylor Series

[2] The following commands evaluate $\sin(\pi/4)$ using three forms of Taylor series (1) expand at $a=0$ to the 5th order, (2) expand at $a=0$ to the 7th order, and (3) expand at $a=\pi/4$ to the 5th order. $\rightarrow$

```
clear
syms x
f(x) = sin(x);
T1(x) = taylor(f)
T2(x) = taylor(f, x, 'Order', 8)
T3(x) = taylor(f, x, pi/4)
vpa(f(pi/4))
vpa(T1(pi/4))
vpa(T2(pi/4))
vpa(T3(pi/4))
```

Table 9.6 Taylor Series

Functions	Description
`expr = taylor(fun)`	Taylor series
`expr = taylor(fun,var,a)`	Taylor series
`expr = taylor(fun,var,name,value)`	Taylor series
Details and More: Help>Symbolic Math Toolbox>Mathematics>Calculus>Series	

Live Editor – Example09_06.mlx *

Example09_06.mlx *

```
clear
syms x
f(x) = sin(x);
T1(x) = taylor(f)
```

T1(x) =

$$\frac{x^5}{120}-\frac{x^3}{6}+x$$

```
T2(x) = taylor(f, x, 'Order', 8)
```

T2(x) =

$$-\frac{x^7}{5040}+\frac{x^5}{120}-\frac{x^3}{6}+x$$

```
T3(x) = taylor(f, x, pi/4)
```

T3(x) =

$$\frac{\sqrt{2}\left(x-\frac{\pi}{4}\right)}{2}+\frac{\sqrt{2}}{2}-\frac{\sqrt{2}\left(x-\frac{\pi}{4}\right)^2}{4}-\frac{\sqrt{2}\left(x-\frac{\pi}{4}\right)^3}{12}+\frac{\sqrt{2}\left(x-\frac{\pi}{4}\right)^4}{48}+\frac{\sqrt{2}\left(x-\frac{\pi}{4}\right)^5}{240}$$

```
vpa(f(pi/4))
```

ans = 0.70710678118654752440084436210485

```
vpa(T1(pi/4))
```

ans = 0.70714304577936024806870707375421

```
vpa(T2(pi/4))
```

ans = 0.70710646957517807081792046856723

```
vpa(T3(pi/4))
```

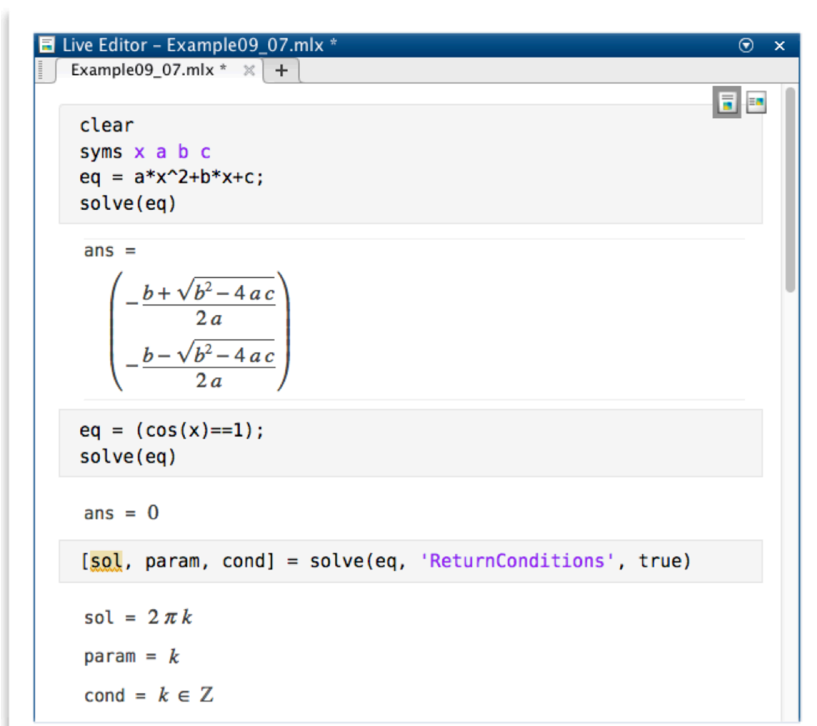
ans = 0.70710678118654752440084436210485

9.7 Algebraic Equations

Example09_07.m: Algebraic Equations

[2] The following commands demonstrate the symbolic solution of algebraic equations.

```
clear
syms x a b c
eq = a*x^2+b*x+c;
solve(eq)
eq = (cos(x)==1);
solve(eq)
[sol, param, cond] = solve(eq, 'ReturnConditions', true)
syms y d e f
eq1 = a*x+b*y==c;
eq2 = d*x+e*y==f;
sol = solve(eq1, eq2)
[sol.x, sol.y]
[A, b] = equationsToMatrix(eq1, eq2, x, y)
sol = linsolve(A, b)
sol = A\b
sol = inv(A)*b
```



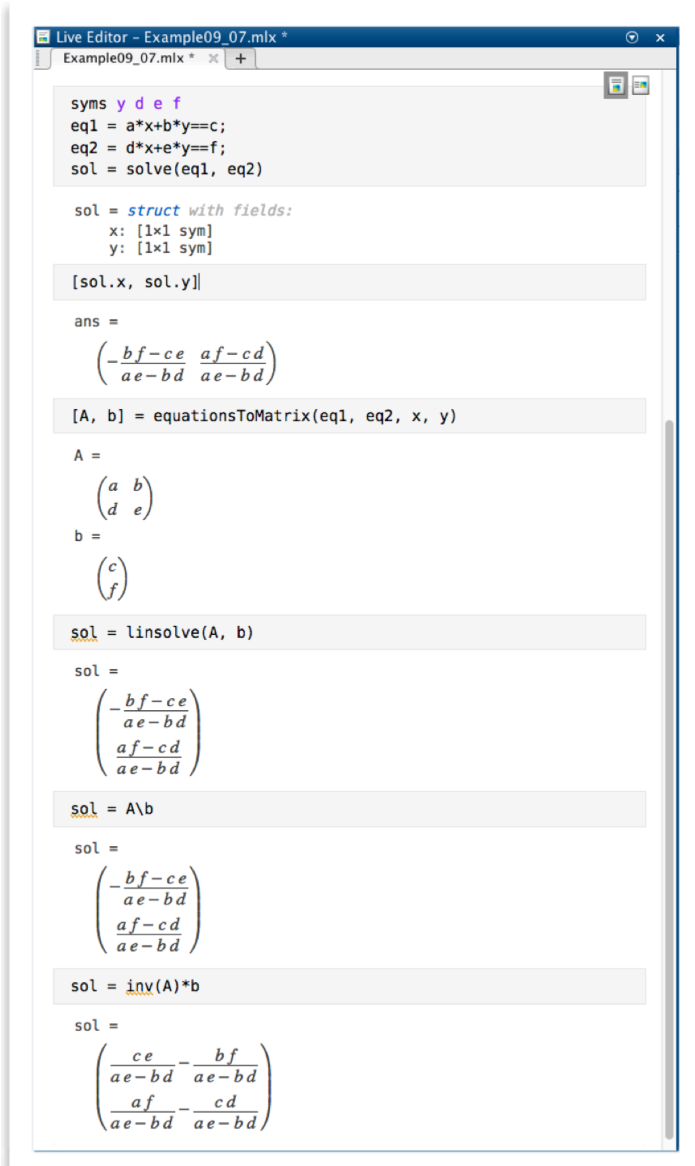


Table 9.7 Algebraic Equations

Functions	Description
`sol = solve(eqn,var)`	Solve algebraic equations
`[A,b] = equationsToMatrix(eqns,vars)`	Convert linear system of equations to matrix form
`sol = linsolve(A,b)`	Solve linear system of equations in matrix form
`x = A\b`	Solve linear system of equations Ax = b
`B = inv(A)`	Inverse of a matrix

Details and More:
Help>Symbolic Math Toolbox>Mathematics>Equation Solving>Linear and Nonlinear Equations and systems

9.8 Inverse of Matrix: Hooke's Law

Example09_08.m: Inverse of Matrix

[2] The following commands calculate **F** defined in [1]. →

```
clear
syms E v G
K = [1/E, -v/E, -v/E,   0,   0    0;
    -v/E,  1/E, -v/E,   0,   0,   0;
    -v/E, -v/E,  1/E,   0,   0,   0;
       0,    0,    0, 1/G,   0,   0;
       0,    0,    0,   0, 1/G,   0;
       0,    0,    0,   0,   0, 1/G]
F = inv(K)
```

Live Editor – Example09_08.mlx *

Example09_08.mlx *

```
clear
syms E v G
K = [1/E, -v/E, -v/E,   0,   0    0;
    -v/E,  1/E, -v/E,   0,   0,   0;
    -v/E, -v/E,  1/E,   0,   0,   0;
       0,    0,    0, 1/G,   0,   0;
       0,    0,    0,   0, 1/G,   0;
       0,    0,    0,   0,   0, 1/G]
```

K =

$$\begin{pmatrix} \frac{1}{\mathrm{E}} & -\frac{v}{\mathrm{E}} & -\frac{v}{\mathrm{E}} & 0 & 0 & 0 \\ -\frac{v}{\mathrm{E}} & \frac{1}{\mathrm{E}} & -\frac{v}{\mathrm{E}} & 0 & 0 & 0 \\ -\frac{v}{\mathrm{E}} & -\frac{v}{\mathrm{E}} & \frac{1}{\mathrm{E}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{G} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{G} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{G} \end{pmatrix}$$

```
F = inv(K)
```

F =

$$\begin{pmatrix} \sigma_2 & \sigma_1 & \sigma_1 & 0 & 0 & 0 \\ \sigma_1 & \sigma_2 & \sigma_1 & 0 & 0 & 0 \\ \sigma_1 & \sigma_1 & \sigma_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & G & 0 & 0 \\ 0 & 0 & 0 & 0 & G & 0 \\ 0 & 0 & 0 & 0 & 0 & G \end{pmatrix}$$

where

$$\sigma_1 = -\frac{\mathrm{E}\,v}{2\,v^2 + v - 1}$$

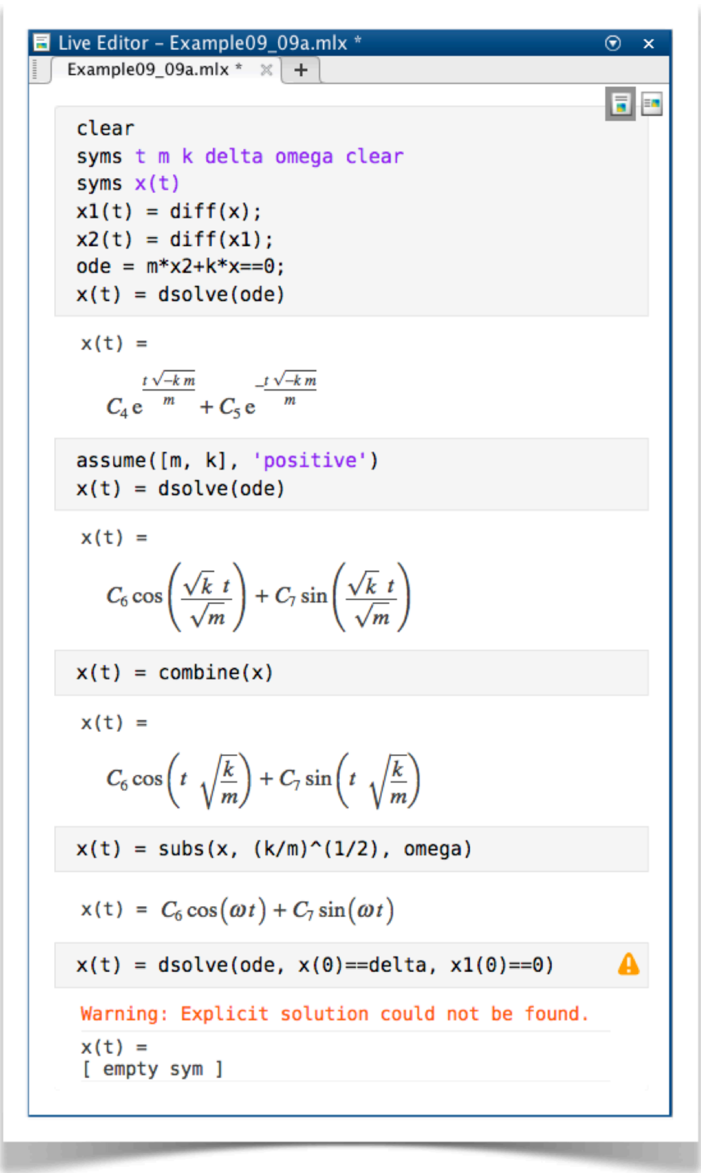
$$\sigma_2 = \frac{\mathrm{E}\,(v-1)}{2\,v^2 + v - 1}$$

9.9 Ordinary Differential Equations (ODE): Vibrations of Supported Machines

Example09_09a.m: Undamped Free Vibrations

[2] The following commands successfully obtain a general solution for the undamped case, Eq. (a); however, they fail to obtain a particular solution that satisfies the initial conditions, Eq. (b). →

```
clear
syms t m k delta omega clear
syms x(t)
x1(t) = diff(x);
x2(t) = diff(x1);
ode = m*x2+k*x==0;
x(t) = dsolve(ode)
assume([m, k], 'positive')
x(t) = dsolve(ode)
x(t) = combine(x)
x(t) = subs(x, (k/m)^(1/2), omega)
x(t) = dsolve(ode, x(0)==delta, x1(0)==0)
```




```
clear
syms t m k delta omega clear
syms x(t)
x1(t) = diff(x);
x2(t) = diff(x1);
ode = m*x2+k*x==0;
x(t) = dsolve(ode)
```

x(t) =

$$C_4\,e^{\frac{t\sqrt{-k\,m}}{m}} + C_5\,e^{-\frac{t\sqrt{-k\,m}}{m}}$$

```
assume([m, k], 'positive')
x(t) = dsolve(ode)
```

x(t) =

$$C_6\cos\left(\frac{\sqrt{k}\;t}{\sqrt{m}}\right) + C_7\sin\left(\frac{\sqrt{k}\;t}{\sqrt{m}}\right)$$

```
x(t) = combine(x)
```

x(t) =

$$C_6\cos\left(t\,\sqrt{\frac{k}{m}}\right) + C_7\sin\left(t\,\sqrt{\frac{k}{m}}\right)$$

```
x(t) = subs(x, (k/m)^(1/2), omega)
```

x(t) = $C_6\cos(\omega t) + C_7\sin(\omega t)$

```
x(t) = dsolve(ode, x(0)==delta, x1(0)==0)
```

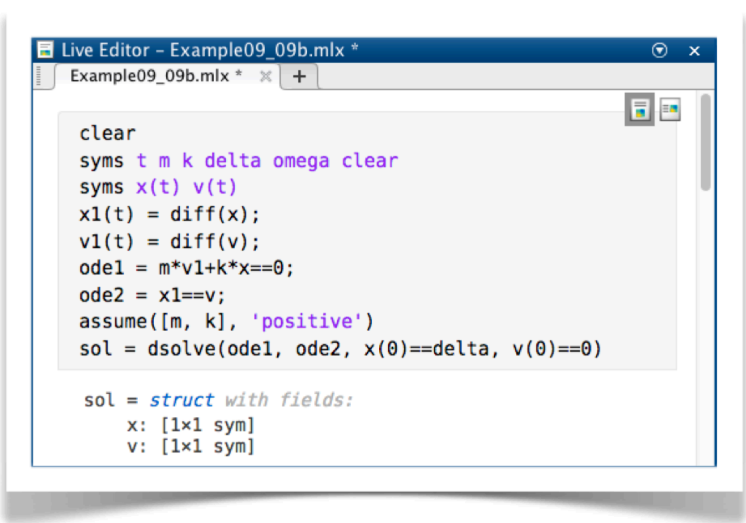
Warning: Explicit solution could not be found.

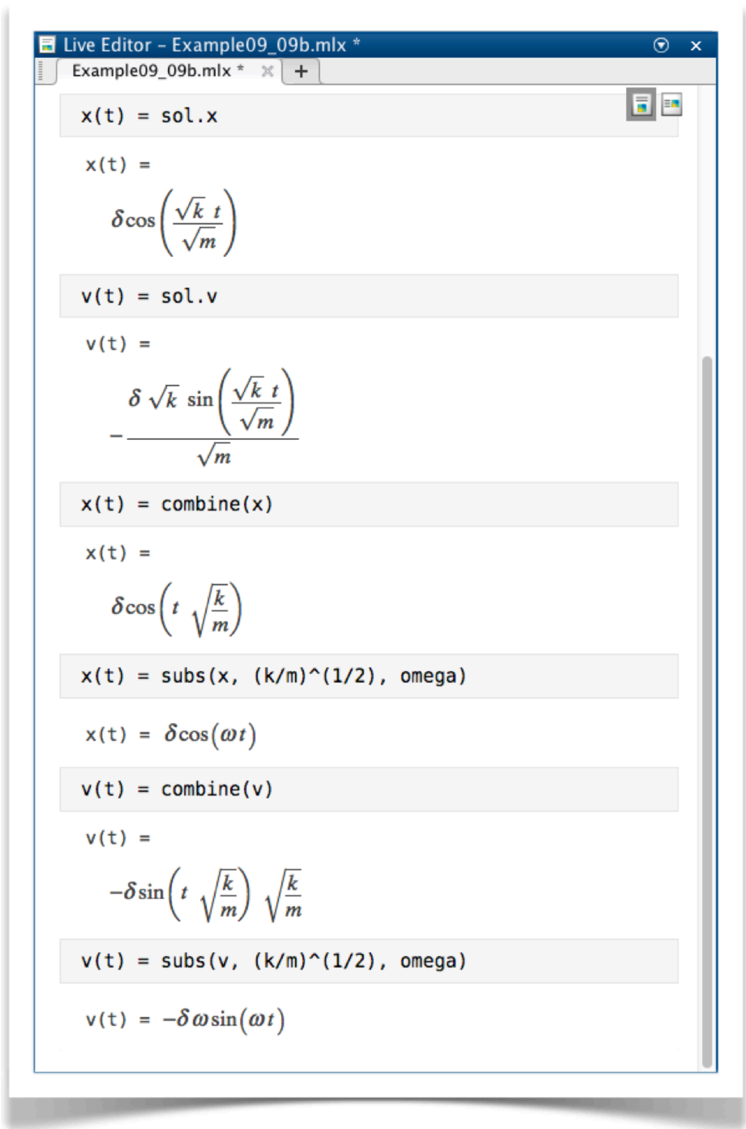
x(t) =
[empty sym]

Example09_09b.m: Undamped Free Vibrations

[6] The following commands successfully obtain a particular solution by solving the system of first-order ODEs, Eqs. (m, n), along with the initial conditions, Eqs. (o, p). →

```
clear
syms t m k delta omega clear
syms x(t) v(t)
x1(t) = diff(x);
v1(t) = diff(v);
ode1 = m*v1+k*x==0;
ode2 = x1==v;
assume([m, k], 'positive')
sol = dsolve(ode1, ode2, x(0)==delta, v(0)==0)
x(t) = sol.x
v(t) = sol.v
x(t) = combine(x)
x(t) = subs(x, (k/m)^(1/2), omega)
v(t) = combine(v)
v(t) = subs(v, (k/m)^(1/2), omega)
```

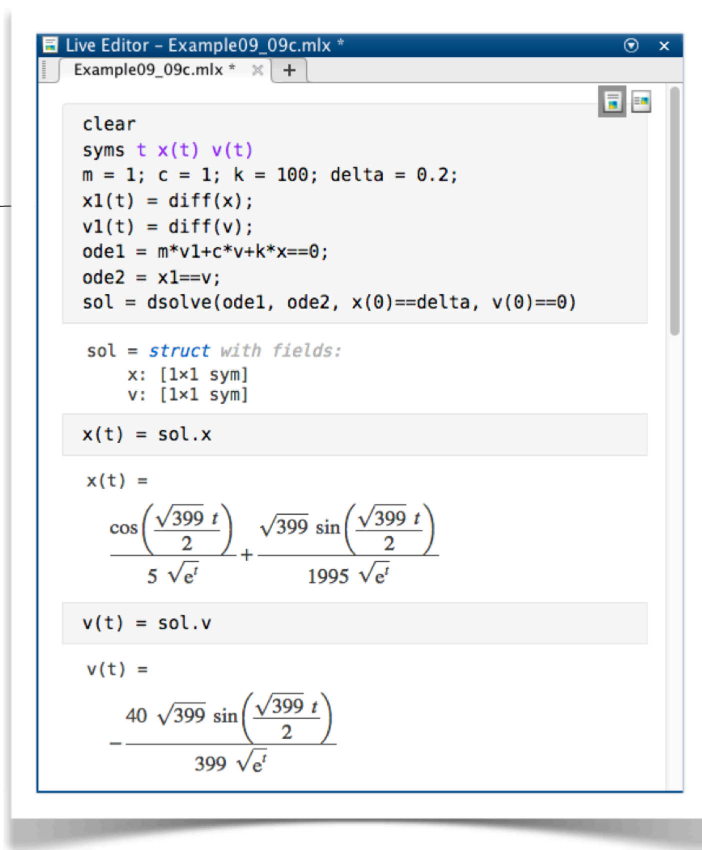



Live Editor – Example09_09b.mlx *
Example09_09b.mlx *
x(t) = sol.x
x(t) =
$\delta\cos\left(\frac{\sqrt{k}\,t}{\sqrt{m}}\right)$
v(t) = sol.v
v(t) =
$-\frac{\delta\,\sqrt{k}\,\sin\left(\frac{\sqrt{k}\,t}{\sqrt{m}}\right)}{\sqrt{m}}$
x(t) = combine(x)
x(t) =
$\delta\cos\left(t\,\sqrt{\frac{k}{m}}\right)$
x(t) = subs(x, (k/m)^(1/2), omega)
x(t) = $\delta\cos(\omega t)$
v(t) = combine(v)
v(t) =
$-\delta\sin\left(t\,\sqrt{\frac{k}{m}}\right)\sqrt{\frac{k}{m}}$
v(t) = subs(v, (k/m)^(1/2), omega)
v(t) = $-\delta\,\omega\sin(\omega t)$

Example09_09c.m: Damped Free Vibrations

[11] The following commands solve the system of ODEs, Eqs. (q, r), along with the initial conditions, Eqs. (s, t).

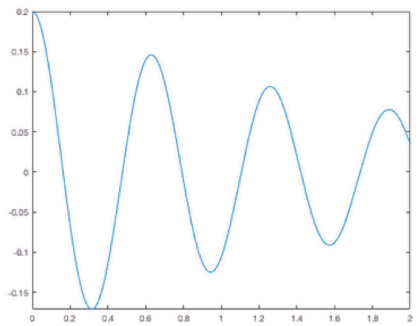
```
clear
syms t x(t) v(t)
m = 1; c = 1; k = 100; delta = 0.2;
x1(t) = diff(x);
v1(t) = diff(v);
ode1 = m*v1+c*v+k*x==0;
ode2 = x1==v;
sol = dsolve(ode1, ode2, x(0)==delta, v(0)==0)
x(t) = sol.x
v(t) = sol.v
fplot(x, [0,2])
fplot(v, [0,2])
vpa(x, 5)
vpa(v, 5)
```



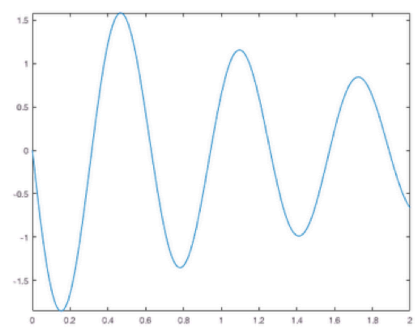
Live Editor – Example09_09c.mlx *

Example09_09c.mlx *

```
fplot(x, [0,2])
```



```
fplot(v, [0,2])
```



```
vpa(x, 5)
```

ans(t) =

$$\frac{0.2\cos(9.9875\,t)}{\sqrt{e^t}}+\frac{0.010013\sin(9.9875\,t)}{\sqrt{e^t}}$$

```
vpa(v, 5)
```

ans(t) =

$$-\frac{2.0025\sin(9.9875\,t)}{\sqrt{e^t}}$$

9.10 Additional Exercise Problems

Chapter 10

Linear Algebra, Polynomial, Curve Fitting, and Interpolation

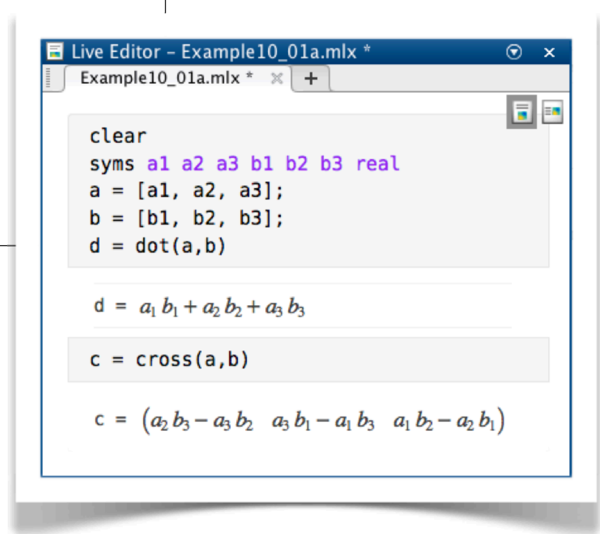
So far, we've introduced the core functionalities of MATLAB. Starting from this chapter, we'll introduce topics that are useful for junior engineering college students. We'll use either symbolic approach or numeric approach whenever it is more instructional and practical.

10.1 Products of Vectors

Example10_01a.m: Products of Vectors

[2] These commands calculate the **dot product** and **cross product** expressed in Eqs. (b) and (d), using the symbolic approach.

```
clear
syms a1 a2 a3 b1 b2 b3 real
a = [a1, a2, a3];
b = [b1, b2, b3];
d = dot(a,b)
c = cross(a,b)
```



Example10_01b.m: Angle and Normal

[6] Given three points, these commands calculate an angle and a unit normal according to [5], using the numerical approach.

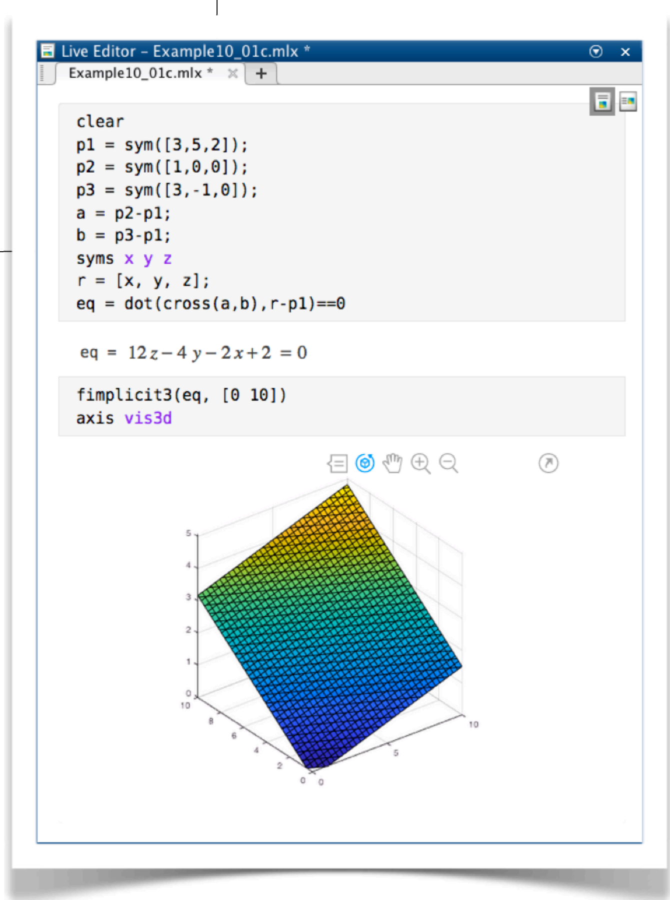
```
clear
p1 = [3, 5, 2];
p2 = [1, 0 ,0];
p3 = [3, -1, 0];
a = p2-p1;
b = p3-p1;
d = dot(a,b)
theta = acosd(d/(norm(a)*norm(b)))
c = cross(a,b)
theta = asind(norm(c)/(norm(a)*norm(b)))
n = c/norm(c)
```

```
>> Example10_01b
d =
    34
theta =
       20.639
c =
    -2    -4    12
theta =
       20.639
n =
     -0.15617     -0.31235      0.93704
>>
```

Example10_01c.m: Plane Defined by 3 Points

[10] Given three points, these commands find the equation of the plane defined by the three points, according to Eq. (j). Knowing the equation, this script then plots the plane.

```
clear
p1 = sym([3,5,2]);
p2 = sym([1,0,0]);
p3 = sym([3,-1,0]);
a = p2-p1;
b = p3-p1;
syms x y z
r = [x, y, z];
eq = dot(cross(a,b),r-p1)==0
fimplicit3(eq, [0 10])
axis vis3d
```



Example10_01d.m: Converting a Symbolic Expression to a Numerical Function

[14] Often, we need to convert a symbolic expression to a MATLAB function, so we may process the data numerically. We now demonstrate the use of the function `matlabFunction` (see 1.7[8], page 30) to convert a symbolic expression to a numerical function. And the plane is plotted using the function `mesh`.

Now, while the symbolic expression `eq` is still in the Workspace, execute the following commands:

```
41  z = solve(eq, z)
42  z = matlabFunction(z)
43  [X,Y] = meshgrid(0:10,0:10);
44  Z = z(X,Y);
45  mesh(X,Y,Z)
```

Line 41 convert the equation to the form $z = z(x,y)$; the result is

$$z = \frac{x}{6} + \frac{y}{3} - \frac{1}{6} \qquad \text{(k)}$$

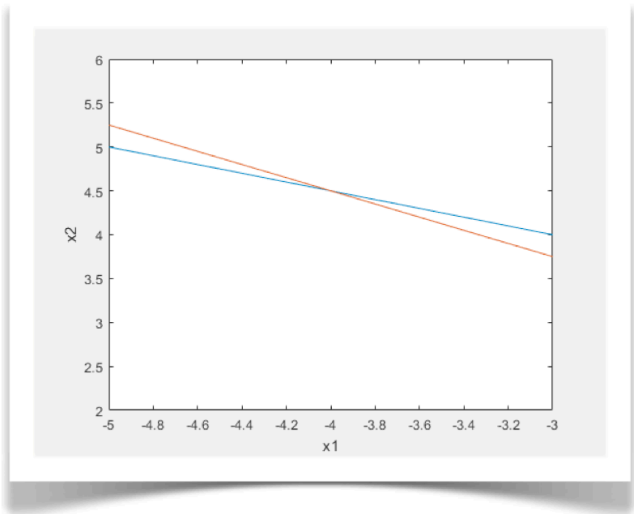
Line 42 converts the symbolic expression $z(x,y)$ into a MATLAB function using `matlabFunction`. Note that the variable `z` before the conversion is of type `sym`.; after the conversion, it is a function handle.

Line 43 creates a mesh grid in the *X*-*Y* space, each axis ranging from 0 to 10. Line 44 calculates *z*-values using the plane equation, Eq. (k). Line 45 plots the plane. #

Table 10.1 Summary of Functions

Functions	Description
`syms v1 v2 ... real`	Symbolic variables
`dot(a, b)`	Dot product
`cross(a, b)`	Cross product
`norm(a)`	Vector and matrix norm
`asind(x)`	Inverse sine in degrees
`acosd(x)`	Inverse cosine in degrees
`fimplicit3(f, interval)`	Plot 3-D implicit function
`solve(eq, z)`	Solve equations
`matlabFunction(expr)`	Convert symbolic expression to MATLAB function
`meshgrid(x,y)`	Generate 2D grid
`mesh(X,Y,Z)`	Mesh plot
`axis vis3d`	Freeze aspect ratio of axes

10.2 Systems of Linear Equations

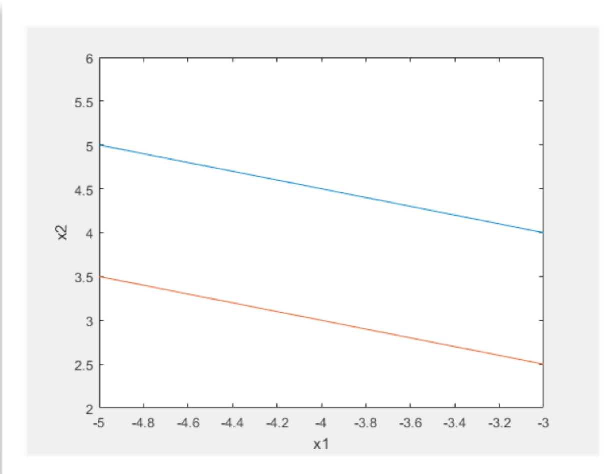


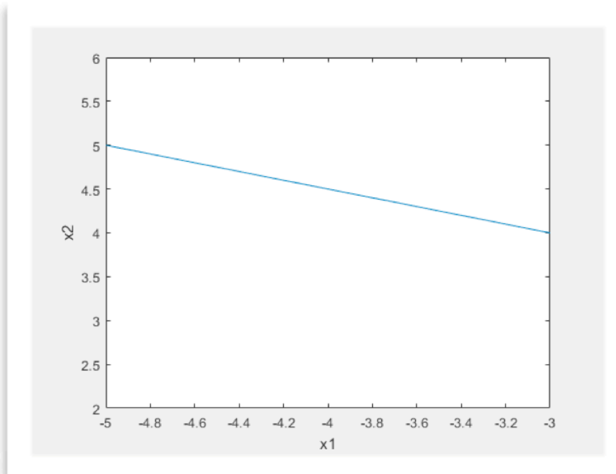
Example10_02a.m: A System of Two Linear Equations

[7] These commands solve a system of linear equations, using the methods described in [2] (last page) and [6]. →

```
clear
A = [1 2; 3 4]; b = [5; 6];
det(A)
x = linsolve(A, b)
x = A\b
A = A'; b = b';
det(A)
x = b/A
A = [1 2; 3 6]; b = [5; 6];
det(A)
x = A\b
b = [5; 15];
x = A\b
```

```
>> Example10_02a
ans =
    -2
x =
   -4.0000
    4.5000
x =
   -4.0000
    4.5000
ans =
    -2
x =
   -4.0000    4.5000
ans =
     0
> In Example10_02a (line 11)
Warning: Matrix is singular to working precision.
x =
  -Inf
   Inf
> In Example10_02a (line 13)
Warning: Matrix is singular to working precision.
x =
   NaN
   NaN
```





Example10_02b.m: Three-Bar Truss

[13] These commands solve the system of linear equations given in Eq. (b) of 3.14[1], page 154.

```
clear
a = sqrt(2)/2;
A = [-a   a   0   0   0   0;
      -a  -a   0   0   0   0;
       a   0   1   1   0   0;
       a   0   0   0   1   0;
       0  -a  -1   0   0   0;
       0   a   0   0   0   1];
b = [0, 1000, 0, 0, 0, 0]';
x = A\b
x = linsolve(A,b)
det(A)
A = A';
b = b';
x = b/A
```

```
>> Example10_02b
x =
  -707.1068
  -707.1068
   500.0000
          0
   500.0000
   500.0000
x =
  -707.1068
  -707.1068
   500.0000
          0
   500.0000
   500.0000
ans =
    -1.0000
x =
  -707.1068 -707.1068   500.0000          0   500.0000   500.0000
```

Table 10.2 Summary of Functions

Functions	Description
`det(A)`	Matrix determinant
`linsolve(A, b)`	Solve linear system of equations
`x = A\b`	Solve linear system of equations
`x = b/A`	Solve linear system of equations
`[L, U] = lu(A)`	LU matrix factorization

10.3 Backslash Operator (\)

```
If A is not sparse
    If A is square
        if A is triangular
            Use triangular solver
        else if A is permuted triangular
            Use permuted triangular solver
        else if A is Hermitian
            If the diagonals of A are real and positive
                Use Cholesky solver
                if Cholesky failed
                    Use LDL solver
                end
            else
                Use LDL solver
            end
        else if A is upper Hessenberg
            Use Hessenberg solver
        else
            Use LU solver
        end
    else (A is not square)
        Use QR solver
    end
else (A is sparse)
    If A is square
        Compute the bandwidth of A
        If A is diagonal
            Use diagonal solver
        else if A is tridiagonal
            Use tridiagonal solver
        else if A is banded
            Use banded solver
        else if A is triangular
            Use triangular solver
        else if A is permuted triangular
            Use permuted triangular solver
        else if A is Hermitian
            if the diagonals of A are real and positive
                Use Cholesky solver
                if Cholesky failed
                    Use LDL solver
                end
            else if A is real
                Use LDL solver
            else
                Use LU solver
            end
        else
            Use LU solver
        end
    else (A is not square)
        Use QR solver
    end
end
```

Example10_03.m: LU Factorization

[5] These commands illustrate the method in [4]. →

```
clear
a = sqrt(2)/2;
A = [-a  a  0  0  0  0;
     -a -a  0  0  0  0;
      a  0  1  1  0  0;
      a  0  0  0  1  0;
      0 -a -1  0  0  0;
      0  a  0  0  0  1];
b = [0, 1000, 0, 0, 0, 0]';
[L,U] = lu(A)
y = L\b
x = U\y
```

```
>> Example10_03
L =
    1.0000         0         0         0         0         0
    1.0000    1.0000         0         0         0         0
   -1.0000   -0.5000    1.0000         0         0         0
   -1.0000   -0.5000         0         0    1.0000         0
         0    0.5000   -1.0000    1.0000         0         0
         0   -0.5000         0         0         0    1.0000
U =
   -0.7071    0.7071         0         0         0         0
         0   -1.4142         0         0         0         0
         0         0    1.0000    1.0000         0         0
         0         0         0    1.0000         0         0
         0         0         0         0    1.0000         0
         0         0         0         0         0    1.0000
y =
           0
        1000
         500
           0
         500
         500
x =
  -707.1068
  -707.1068
   500.0000
          0
   500.0000
   500.0000
```

10.4 Eigenvalue Problems

Example10_04.m: Eigenvalue Problems

[2] These commands solve the eigenvalue problem described in Eqs. (d-f), last page.

```
clear
K = [40, -25,   0;
    -25,  50, -30;
      0, -30,  60]
M = diag([3, 4, 5])
[X, Lamda] = eig(K, M)
Omega = sqrt(Lamda)
```

```
>> Example10_04
K =
    40   -25     0
   -25    50   -30
     0   -30    60
M =
     3     0     0
     0     4     0
     0     0     5
X =
   -0.2788    0.3918   -0.3195
   -0.3547    0.0337    0.3508
   -0.2296   -0.3271   -0.2008
Lamda =
    2.7318         0         0
         0   12.6175         0
         0         0   22.4840
Omega =
    1.6528         0         0
         0    3.5521         0
         0         0    4.7417
```

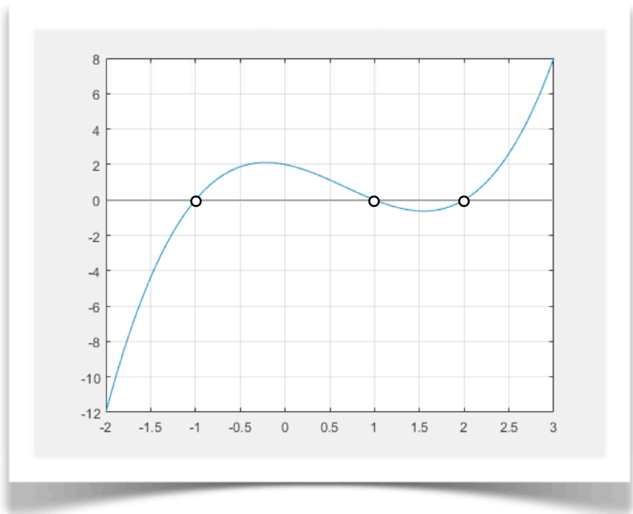
10.5 Polynomials

Example10_05a.m: Polynomials

[2] These commands demonstrate some basic functions of manipulating the MATLAB polynomials.

```
clear
p = [1, -2, -1, 2];
polyval(p, 3)
x = linspace(-2,3);
plot(x, polyval(p,x)), grid on
r = roots(p)
poly(r)
p1 = polyint(p)
polyder(p1)
a = [1, 3, 5];
b = [2, 4, 6];
c = polyder(a,b)
[n,d] = polyder(a,b)
```

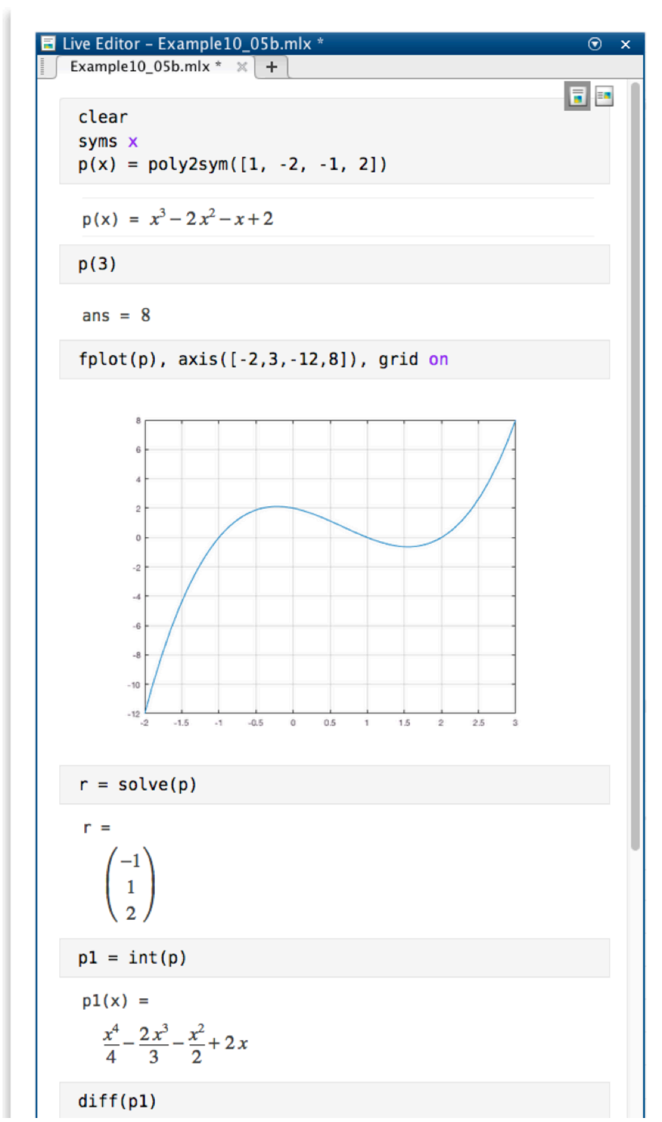
```
>> Example10_05a
ans =
     8
r =
   -1.0000
    2.0000
    1.0000
ans =
    1.0000   -2.0000   -1.0000    2.0000
p1 =
    0.2500   -0.6667   -0.5000    2.0000         0
ans =
     1    -2    -1     2
c =
     8    30    56    38
n =
    -2    -8    -2
d =
     4    16    40    48    36
```


8
6
4
2
0
-2
-4
-6
-8
-10
-12
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3

Example10_05b.m: Symbolic Polynomials

[6] These commands do the same tasks as those in [2], but using symbolic mathematics instead.

```
clear
syms x
p(x) = poly2sym([1, -2, -1, 2])
p(3)
fplot(p), axis([-2,3,-12,8]), grid on
r = solve(p)
p1 = int(p)
diff(p1)
a = poly2sym([1, 3, 5])
b = poly2sym([2, 4, 6])
c = diff(a*b)
[n,d] = numden(diff(a/b))
```



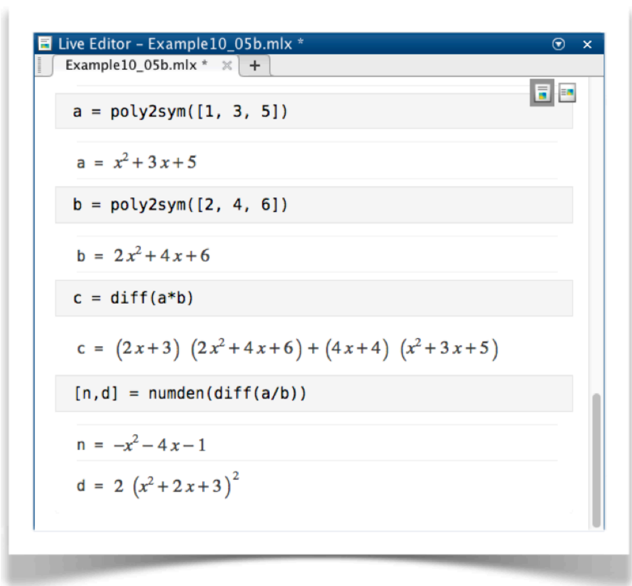


Table 10.5 Summary of Functions

Functions	Description
`polyval(p, x)`	Evaluates a polynomial p at x
`roots(p)`	Finds the roots of the equation p = 0
`poly(r)`	Returns a polynomial of specified roots r
`polyint(p)`	Integrates a polynomial p
`polyder(p)`	Differentiates a polynomial p
`p = polyder(p1,p2)`	Differentiates the product of polynomials p1 and p2
`[n,d] = polyder(p1,p2)`	Differentiates the division of polynomials p1 and p2
`s = poly2sym(p)`	Converts a polynomial from numeric form to symbolic form
`sym2poly(expr)`	Converts a polynomial from symbolic form to numeric form
`int(expr)`	Integrates a symbolic expression
`diff(expr)`	Differentiates a symbolic expression
`[n,d] = numden(expr)`	Extracts numerator and denominator of a symbolic expression

10.6 Polynomial Curve Fittings

Temperature T (K)	33	144	255	366	477	589	700	811	922
Young's Modulus E (GPa)	220	213	206	199	192	185	167	141	105

Example10_06.m: Polynomial Curve Fittings

[2] This script finds the polynomials of degrees one, two, and three that best-fit the data points in [1]. →

```
clear
T = [ 33, 144, 255, 366, 477, 589, 700, 811, 922];
E = [220, 213, 206, 199, 192, 185, 167, 141, 105];
temp = 0:50:1000;
format shortG
for k = 1:3
    P = polyfit(T, E, k)
    young = polyval(P, temp);
    subplot(2,2,k)
    plot(T, E, 'o', temp, young)
    axis([0,1000,100,250])
    xlabel('Temperature (K)')
    ylabel('Young''s Modulus (GPa)')
    title(['Polynomial of Degree ', num2str(k)])
    residual = E - polyval(P, T);
    error = norm(residual)
    R = sqrt(1-error^2/norm(E)^2)
end
```

```
>> Example10_06
P =
     -0.11514       235.86
error =
       37.055
R =
      0.99775
P =
  -0.00015541     0.033291       213.23
error =
       15.469
R =
      0.99961
P =
  -2.9124e-07   0.00026175     -0.12346       224.67
error =
       3.3894
R =
      0.99998
```

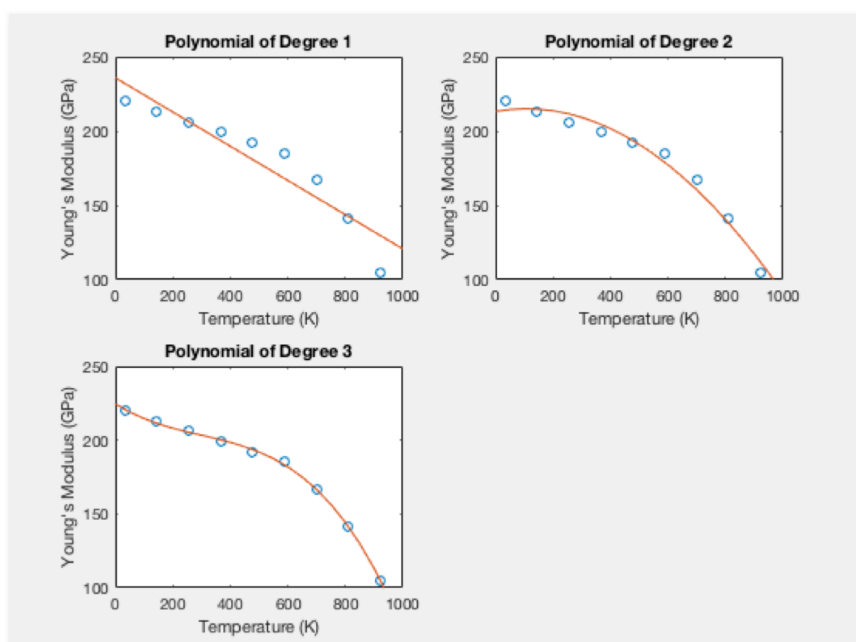
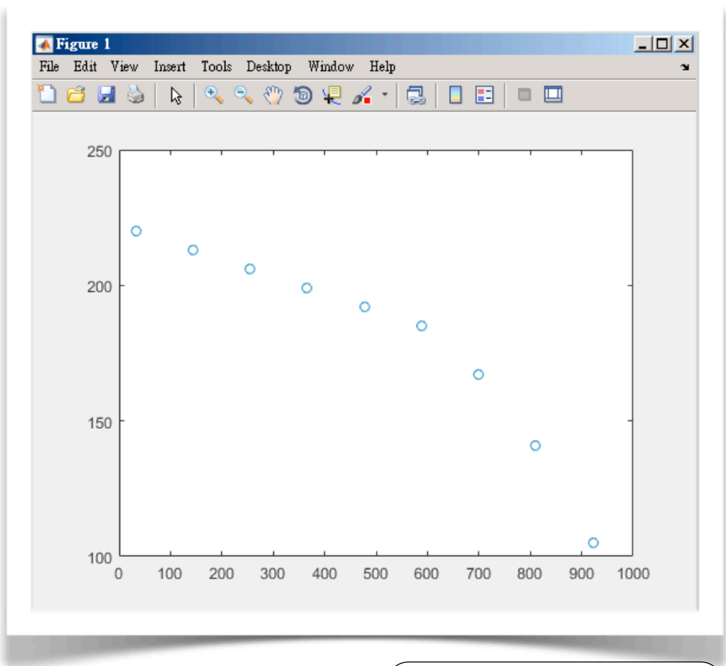


Table 10.6 Summary of Functions	
Functions	Description
`format shortG`	Uses `short` or `shortE`, whichever is more compact
`polyfit(x,y,n)`	Finds a polynomial of degree n that is a best fit for the data points (x, y)
`polyval(p,x)`	Evaluates a polynomial p at x
`subplot(m,n,k)`	Creates axes in tiled positions
`norm(a)`	Calculates vector/matrix norm

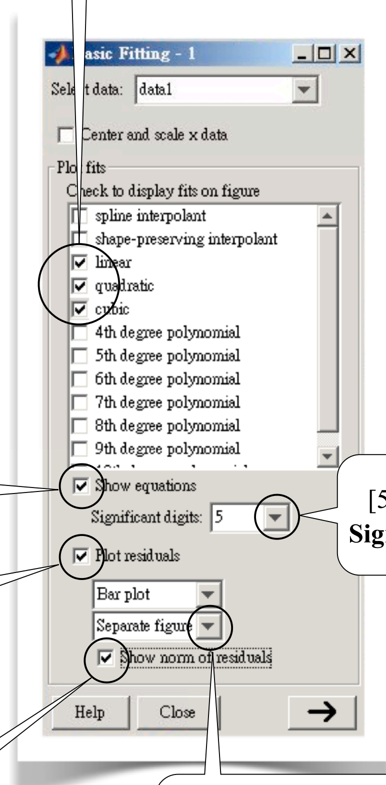
10.7 Interactive Curve-Fitting Tools

[1] In this section, we'll demonstrate the use of an interactive curve-fitting tool built in MATLAB to perform the tasks covered in the last section. First, prepare the data points and create a figure:

```
>> clear
>> T = [ 33, 144, 255, 366, 477, 589, 700, 811, 922];
>> E = [220, 213, 206, 199, 192, 185, 167, 141, 105];
>> plot(T, E, 'o')
>> axis([0, 1000, 100, 250])
```



[3] Select **linear**, **quadratic**, and **cubic** (see [9], next page).



[4] Select **Show equations** (see [10], next page)

[5] Select **5** for **Significant digits**.

[6] Select **Plot residuals**.

[8] Select **Show norm of residuals** (see [12], next page). →

[7] Select **Separate figure** (see [11], next page).

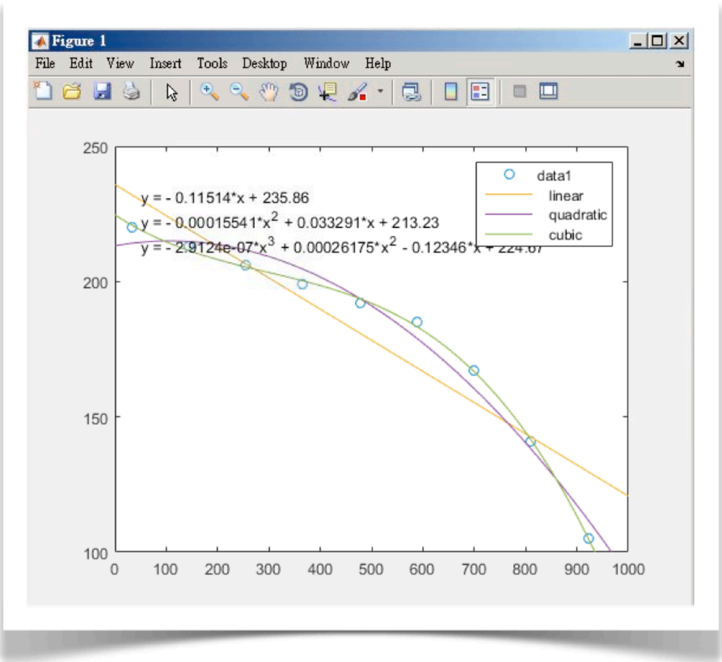


Figure 1
File Edit View Insert Tools Desktop Window Help
data1
linear
quadratic
cubic
y = - 0.11514*x + 235.86
y = - 0.00015541*x^2 + 0.033291*x + 213.23
y = - 2.9124e-07*x^3 + 0.00026175*x^2 - 0.12346*x + 224.67
250
200
150
100
0
100
200
300
400
500
600
700
800
900
1000

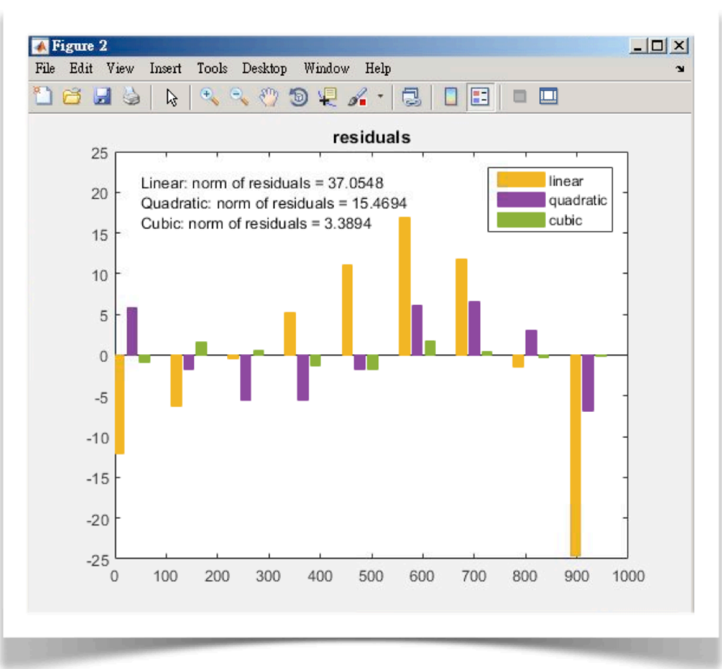
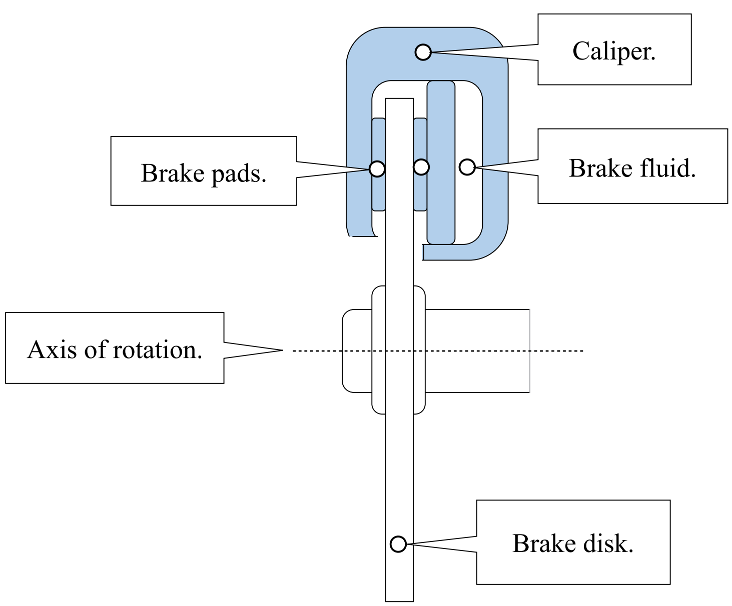
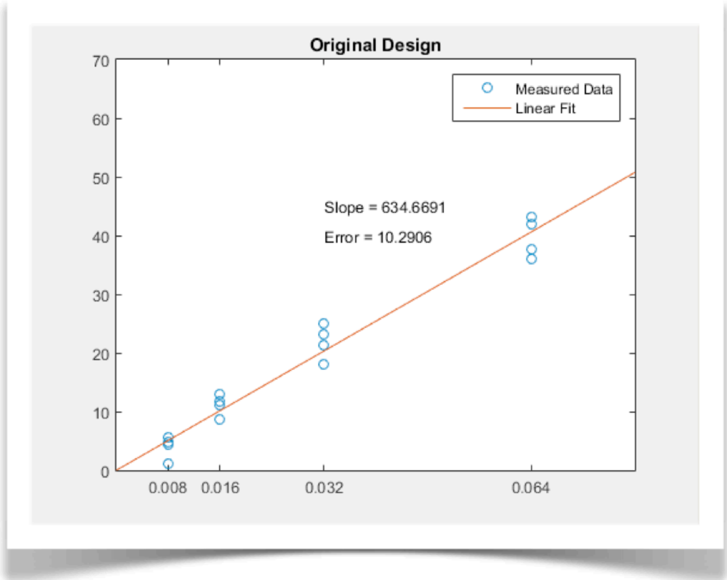


Figure 2
File Edit View Insert Tools Desktop Window Help
residuals
Linear: norm of residuals = 37.0548
Quadratic: norm of residuals = 15.4694
Cubic: norm of residuals = 3.3894
linear
quadratic
cubic
25
20
15
10
5
0
-5
-10
-15
-20
-25
0
100
200
300
400
500
600
700
800
900
1000

10.8 Linear Fit Through Origin: Brake Assembly

Brake fluid pressure x (kgf/mm^2)	0.008	0.016	0.032	0.064
Braking torque y (kgf-mm)	4.8	11.1	23.1	42.0
	1.2	8.6	18.1	36.0
	5.7	13.0	25.1	43.2
	4.4	11.8	21.4	37.6




Original Design
Measured Data
Linear Fit
Slope = 634.6691
Error = 10.2906
70
60
50
40
30
20
10
0
0.008
0.016
0.032
0.064

Example10_08a.m: The Original Design

[4] This script finds a line through origin that best-fits the data points in [1] and calculates the **error**, Eq. (d).

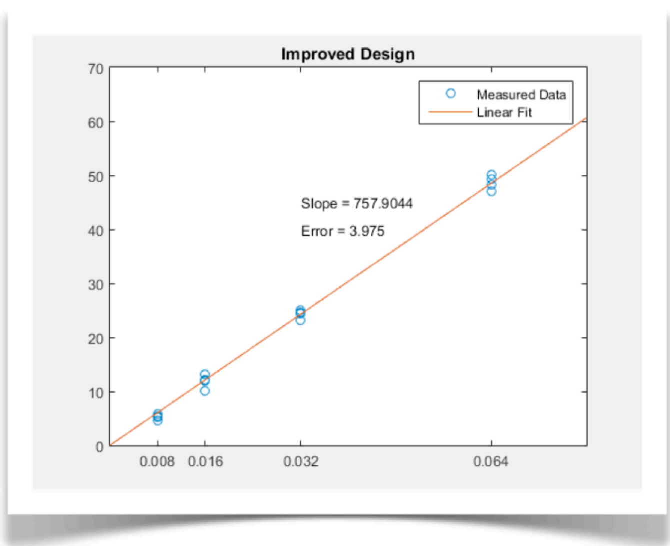
```
clear
x = [0.008, 0.008, 0.008, 0.008, ...
     0.016, 0.016, 0.016, 0.016, ...
     0.032, 0.032, 0.032, 0.032, ...
     0.064, 0.064, 0.064, 0.064];
y = [4.8,  1.2,  5.7,  4.4, ...
    11.1,  8.6, 13.0, 11.8, ...
    23.1, 18.1, 25.1, 21.4, ...
    42.0, 36.0, 43.2, 37.6];
slope = sum(x.*y)/sum(x.^2);
residual = y-slope*x;
error = norm(residual);
plot(x,y,'o'), hold on
hAxes = gca;
hAxes.XTick = [0.008,0.016,0.032,0.064];
axis([0,0.08,0,70]);
plot([0,0.08],[0,slope*0.08])
text(0.032, 45, ['Slope = ', num2str(slope)])
text(0.032, 40, ['Error = ', num2str(error)])
legend('Measured Data', 'Linear Fit')
title('Original Design')
```

Brake fluid pressure x (kgf/mm^2)	0.008	0.016	0.032	0.064
Braking torque y (kgf-mm)	5.3	12.2	24.6	49.3
	4.6	10.1	23.1	47.1
	5.8	13.2	25.0	50.1
	5.4	11.9	24.3	48.2

Example10_08b.m: An Improved Design

[7] This script finds a line through origin that best-fits the data points listed in [6], last page, and calculates the **error**. The dimmed lines are copied from Example10_08a.m, last page; the other statements are self-explained.

```
clear
x = [0.008, 0.008, 0.008, 0.008, ...
     0.016, 0.016, 0.016, 0.016, ...
     0.032, 0.032, 0.032, 0.032, ...
     0.064, 0.064, 0.064, 0.064];
y = [5.3,  4.6,  5.8,  5.4, ...
    12.2, 10.1, 13.2, 11.9, ...
    24.6, 23.1, 25.0, 24.3, ...
    49.3, 47.1, 50.1, 48.2];
slope = sum(x.*y)/sum(x.^2);
residual = y-slope*x;
error = norm(residual);
plot(x,y,'o'), hold on
hAxes = gca;
hAxes.XTick = [0.008,0.016,0.032,0.064];
axis([0,0.08,0,70]);
plot([0,0.08],[0,slope*0.08])
text(0.032, 45, ['Slope = ', num2str(slope)])
text(0.032, 40, ['Error = ', num2str(error)])
legend('Measured Data', 'Linear Fit')
title('Improved Design')
```



Using Backslash Operator for Least Squares Linear Fit

[9] In Section 10.3, when discussing systems of linear equations

$$\mathbf{Ax} = \mathbf{b}$$

We assumed that **A** is an n-by-n square matrix, **x** is an n-by-1 vector, and **b** is also an n-by-1 vector. (We also assume the determinant of **A** is nonzero.)

What if the system of equations is over-specified; i.e., the number of equations is more than the number of unknown variables? Assume that **A** is an m-by-n matrix (**x** is n-by-1 and **b** is m-by-1), where $m>n$. In such cases, the backslash operator **A\b** will use a least-squares fit algorithm similar to that used by `polyfit`, discussed in Section 10.6.

Now, returning to our problem of finding a line $y = bx$ that best fits n points (x_k, y_k), $k = 1, 2, \ldots n$, we may treat the problem as

$$\mathbf{x}'b = \mathbf{y}'$$

where $\mathbf{x}'$ is n-by-1, b is 1-by-1, and $\mathbf{y}'$ is n-by-1. The least squares solution of b can be calculated with

```
b = x'\y'
```

To confirm these concepts, at the end of Example10_08b.m, type

```
>> format short
>> b = x'\y'
b =
  757.9044
```

Which are consistent with that calculated using Eq. (c) (page 413) and shown in [8] (last page).

The built-in function `lsqlin` (least-square linear regression) also does the same job:

```
>> b = lsqlin(x',y')
b =
  757.9044
```

We'll discuss more about `lsqlin` for multivariate linear regression in Section 12.8. #

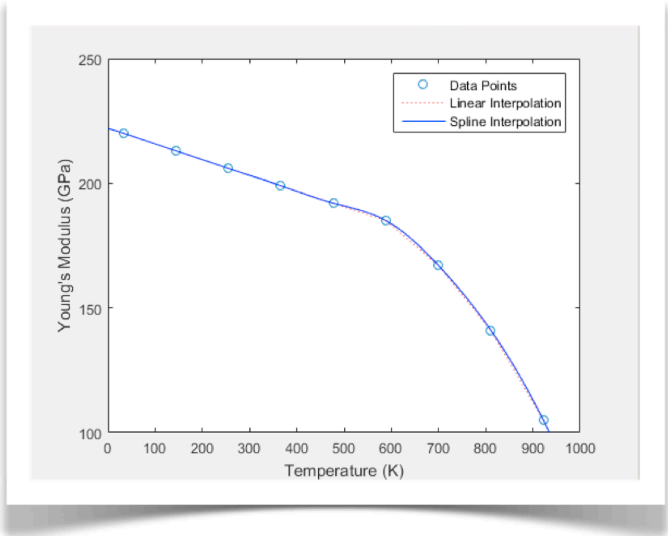
10.9 Interpolations

Temperature T (K)	33	144	255	366	477	589	700	811	922
Young's Modulus E (GPa)	220	213	206	199	192	185	167	141	105

Example10_09a.m: Interpolations

[3] This script produces a graph shown in [4], next page. →

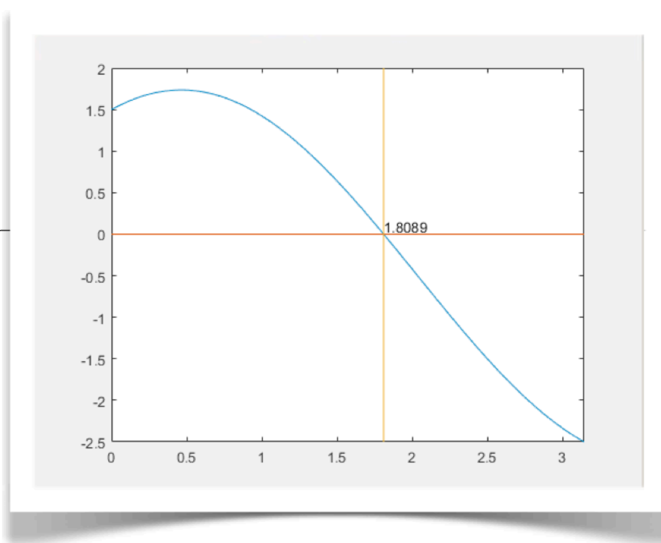
```
clear
T = [ 33, 144, 255, 366, 477, 589, 700, 811, 922];
E = [220, 213, 206, 199, 192, 185, 167, 141, 105];
plot(T, E, 'o'), hold on
temp = 0:10:1000;
young = interp1(T, E, temp);
plot(temp, young, 'r:')
young = interp1(T, E, temp, 'spline');
plot(temp, young, 'b-')
axis([0,1000,100,250])
xlabel('Temperature (K)')
ylabel('Young''s Modulus (GPa)')
legend('Data Points', 'Linear Interpolation', 'Spline Interpolation')
```



Example10_09b.m: Roots Finding by Interpolations

[6] An application of interpolation is to find the roots of an equation. As an example, this script solves the equation $\sin x + 2\cos x - 0.5 = 0$. The output (see [7]) shows that a solution of the equation is $x = 1.8089$.

```
clear
x = linspace(0,pi);
y = sin(x)+2*cos(x)-0.5;
plot(x,y,[0,pi], [0,0]), hold on
axis([0, pi, -2.5, 2])
x1 = interp1(y,x,0)
plot([x1, x1], [-2.5, 2])
text(x1,0.1,num2str(x1))
```

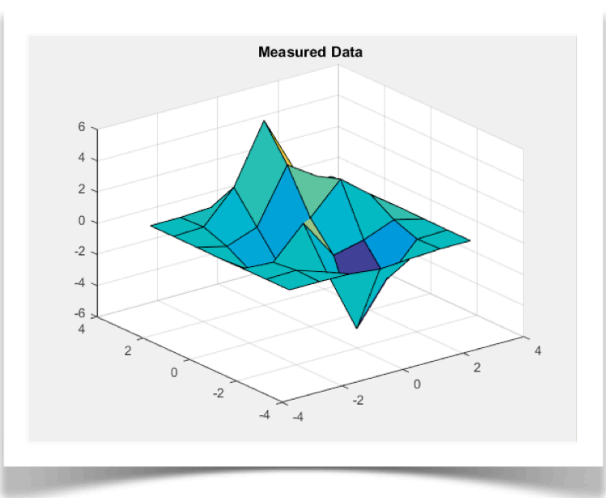


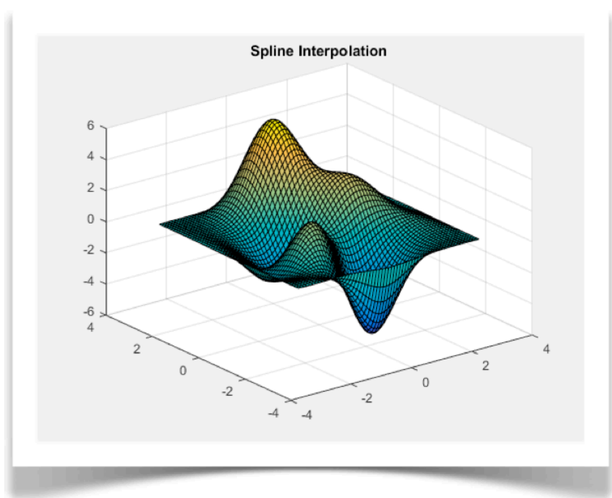
10.10 Two-Dimensional Interpolations

Example10_10.m: Two-Dimensional Interpolations

[2] This script produces two figures as shown in [3, 4].

```
clear
[X,Y] = meshgrid(-3:3);
Z = peaks(X,Y);
surf(X,Y,Z)
title('Measured Data')
[X1,Y1] = meshgrid(-3:0.1:3);
Z1 = interp2(X,Y,Z,X1,Y1,'spline');
figure
surf(X1,Y1,Z1)
title('Spline Interpolation')
```





Chapter 11

Differentiation, Integration, and Differential Equations

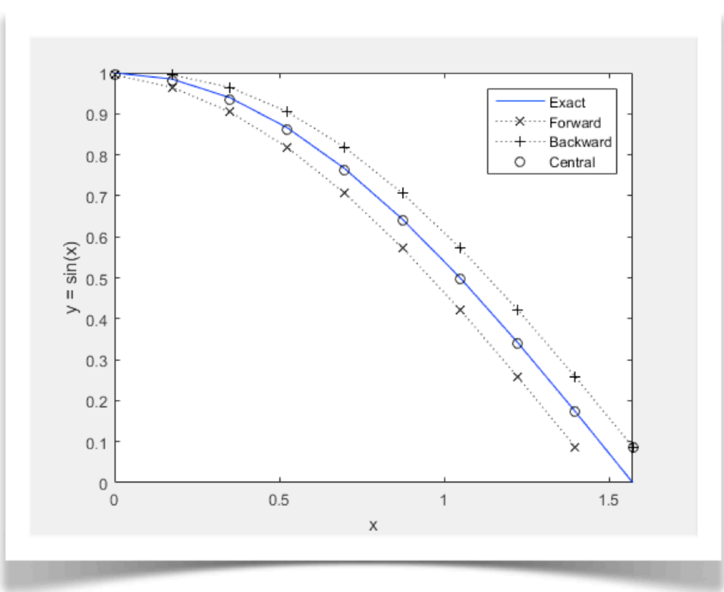
Engineering problem-solving often involves three stages: establishing differential equations for the problem, solving the differential equations, and interpreting the solution. Traditional engineering mathematics is abstract and hard for a junior college student. However, through the use of MATLAB, these mathematics become concrete and easy to understand.

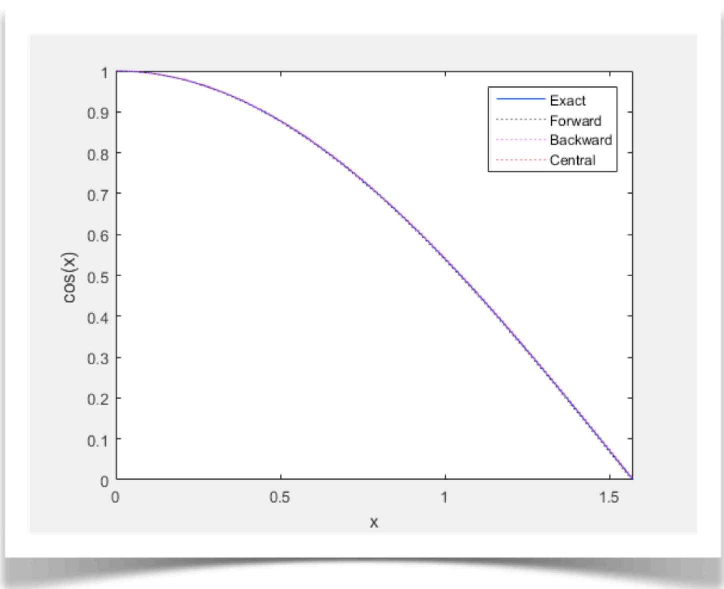
11.1 Numerical Differentiation

Example11_01.m: Forward, Backward, and Central Differences

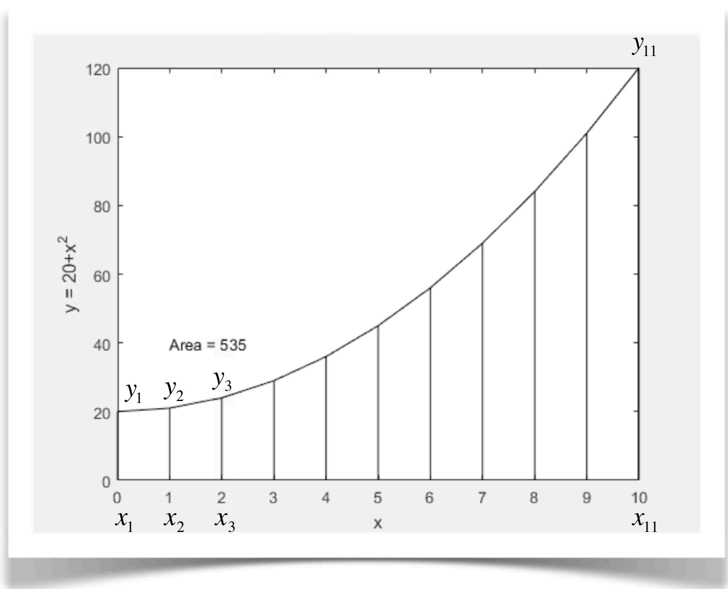
[3] Using $y = \sin(x)$, i.e., $y' = \cos(x)$, as an example, this script compares three numerical differentiation methods introduced in [2], last page. The graphics output is shown in [4-9].

```
clear
n = 10;
x = linspace(0, pi/2, n);
y = sin(x);
y1 = cos(x);
plot(x, y1, 'b-'), hold on
axis([0, pi/2, 0, 1])
y1 = diff(y)./diff(x);
plot(x(1:n-1), y1, 'kx:')
plot(x(2:n), y1, 'k+:')
y1 = gradient(y, x);
plot(x, y1, 'ko')
xlabel('x')
ylabel('cos(x)')
legend('Exact', 'Forward', 'Backward', 'Central')
```




Exact
Forward
Backward
Central
cos(x)
x
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5

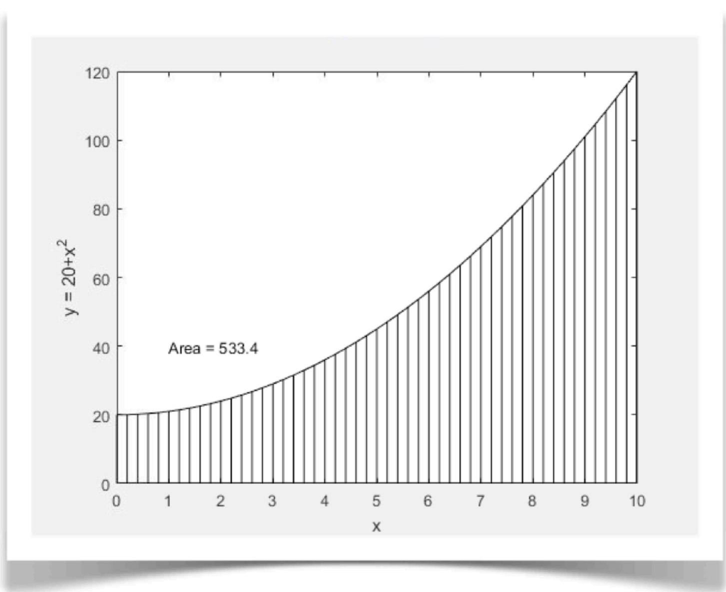
11.2 Numerical Integration: `trapz`



Example11_02.m: Numerical Integration

[3] This script generates a graph as shown in [2], last page, and calculates the area under the curve $y(x) = 20 + x^2$, i.e., performing numerical integration $\int_0^{10} y(x)dx$, using the function `trapz`.

```
clear
n = 11;
x = linspace(0,10,n);
y = 20 + x.^2;
plot(x, y, 'k'), hold on
plot([x;x], [zeros(1,n);y], 'k')
axis([0, 10, 0, 120])
xlabel('x')
ylabel('y = 20+x^2')
A = trapz(x, y)
text(1, 40, ['Area = ', num2str(A)])
```

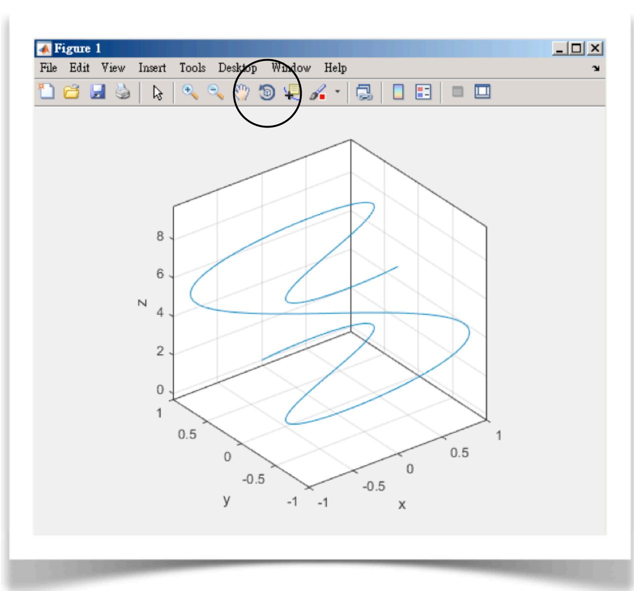


11.3 Length of a Curve

Example11_03a.m: Length of a Curve

[2] This script calculates the length of the curve defined by Eq. (b), using the function `trapz`.

```
clear
n = 500;
t = linspace(0,3*pi,n);
x = sin(2*t);
y = cos(t);
z = t;
plot3(x, y, z)
axis vis3d, box on, grid on
xlabel('x'), ylabel('y'), zlabel('z')
f = sqrt(gradient(x,t).^2+gradient(y,t).^2+gradient(z,t).^2);
L = trapz(t, f)
```

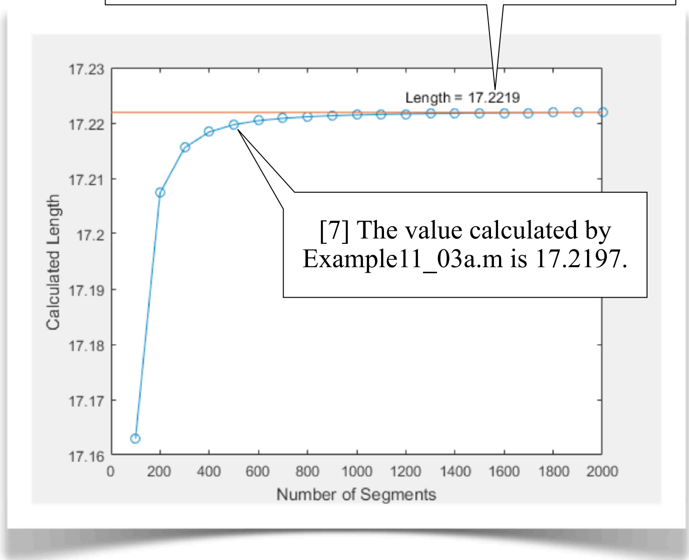


Example11_03b.m: Convergence Study

[5] This script repeats the calculation of the length with $n = 100,\ 200,\ \ldots,\ 2000$, and generates a **convergence curve** as shown in [6-7]. →

```
clear
k = 0;
for n = 100:100:2000;
    t = linspace(0,3*pi,n);
    x = sin(2*t);
    y = cos(t);
    z = t;
    f = sqrt(gradient(x,t).^2+gradient(y,t).^2+gradient(z,t).^2);
    k = k+1;
    L(k) = trapz(t, f);
end
plot([100:100:2000], L, 'o-'), hold on
plot([0,2000], [L(k), L(k)])
text(1200,L(k)+0.003, ['Length = ', num2str(L(k))])
xlabel('Number of Segments')
ylabel('Calculated Length')
```

[6] As the number of segments increases, the **convergence curve** (the curve with circular marks) approaches a horizontal asymptote, of which the *y*-value is the analytical value (exact value).



[7] The value calculated by Example11_03a.m is 17.2197.

11.4 User-Defined Function as Input Argument: `integral`

Example11_04.m: Numerical Integration

[2] This script calculates the length of the curve defined in Eq. (b), page 427, using the function `integral`.

```
L = integral(@fun, 0, 3*pi)

function dL = fun(t)
x = sin(2*t);
y = cos(t);
z = t;
dL = sqrt(gradient(x,t).^2+gradient(y,t).^2+gradient(z,t).^2);
end
```

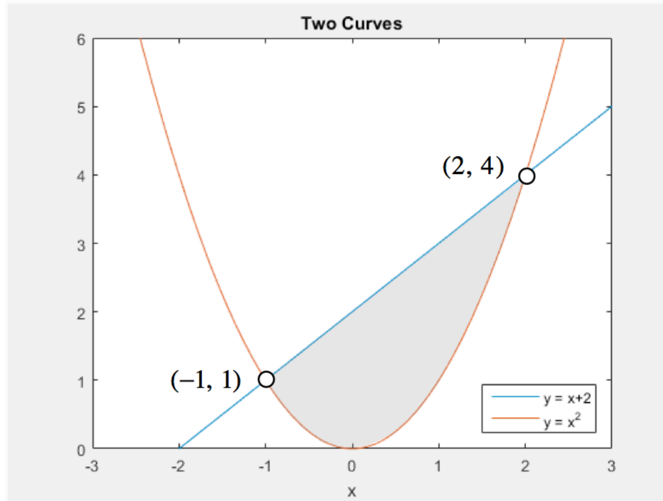
11.5 Area and Centroid

Example11_05.m: Area and Centroid

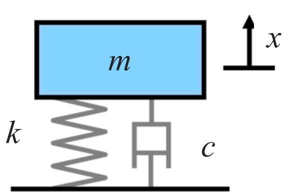
[2] This script calculates the area and the centroid of the area bounded by two curves defined in Eq. (d). →

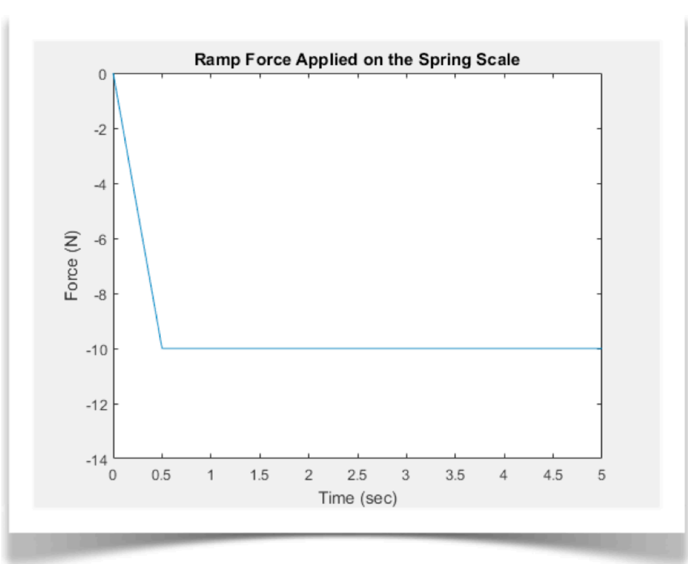
```
clear
syms x y
y1 = x+2;
y2 = x^2;
fplot(y1), hold on, fplot(y2)
axis([-3,3,0,6])
legend('y = x+2', 'y = x^2', 'Location', 'southeast')
title('Two Curves')
eq1 = (y == y1);
eq2 = (y == y2);
sol = solve(eq1, eq2);
x1 = double(sol.x(1))
x2 = double(sol.x(2))
y1 = double(sol.y(1))
y2 = double(sol.y(2))
funA = @(x) abs((x+2)-(x.^2));
A = integral(funA,x1,x2)
funX = @(x) x.*abs((x+2)-(x.^2));
funY = @(x) abs((x+2).^2-(x.^4));
xc = integral(funX,-1,2)/A
yc = integral(funY,-1,2)/(2*A)
```

```
>> Example11_05
x1 =
       -1
x2 =
        2
y1 =
        1
y2 =
        4
A =
        4.5000
xc =
        0.5000
yc =
        1.6000
```



11.6 Placing Weight on Spring Scale

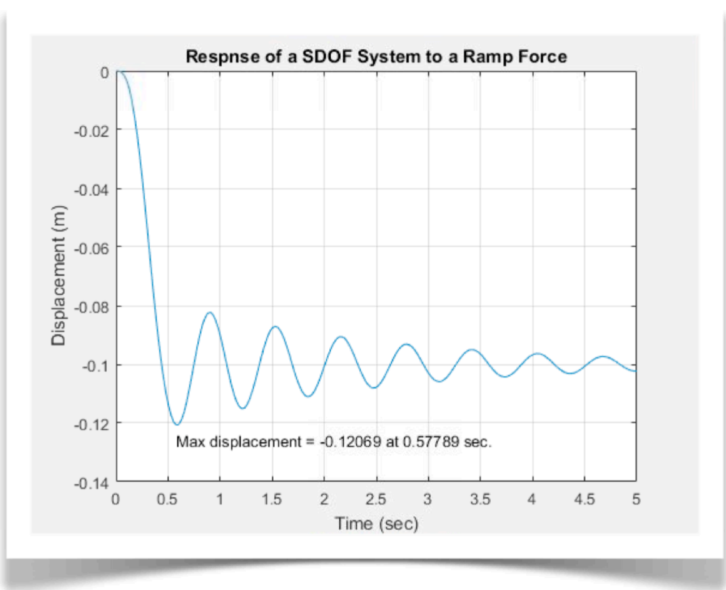




Example11_06.m: Spring Scale

[3] This program calculates the response of the system described in [1], last page.

```
springScale

function springScale
m = 1; c = 1; k = 100; W = -10; T0 = 0.5;
omega = sqrt(k/m); cc = 2*m*omega; zeta = c/cc; t0 = omega*T0;
n = 200; t = linspace(0, 50, n);
for i = 1:n
    x(i) = (W/k)*(h(t(i))*(t(i)>0)-h(t(i)-t0)*(t(i)>t0));
end
plot(t/omega, x), grid on
xlabel('Time (sec)'), ylabel('Displacement (m)')
title('Respnse of a SDOF System to a Ramp Force')
[M,I] = min(x);
text(t(I)/omega, M-0.005, ['Max displacement = ', ...
    num2str(M), ' at ', num2str(t(I)/omega), ' sec.'])

    function out = h(t)
        out = (1/(t0*sqrt(1-zeta^2)))*integral(@fun, 0, t);

        function out = fun(tau)
            out = tau.*exp(-zeta*(t-tau)).*sin((t-tau)*sqrt(1-zeta^2));
        end
    end
end
```

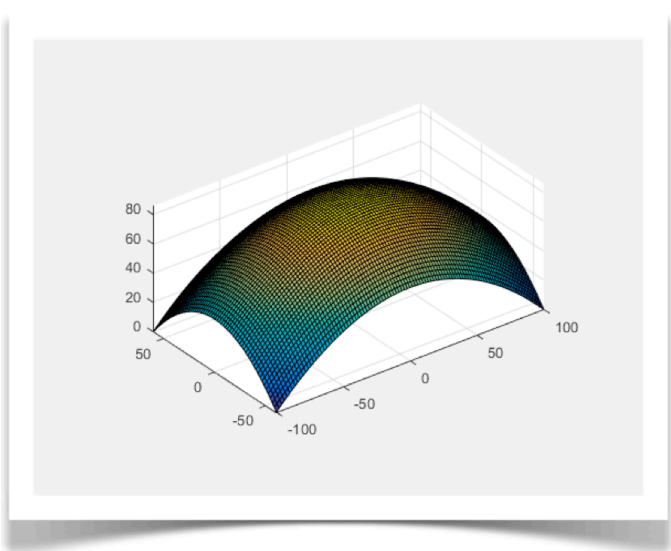


11.7 Double Integral: Volume Under Stadium Dome

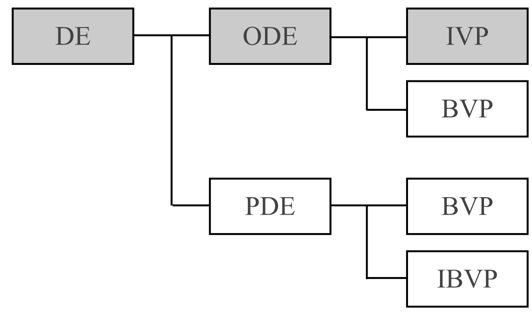
Example11_07.m: Volume of Stadium Dome

[2] This script calculates the volume under the stadium dome described in [1].

```
clear
x = -100:2:100;
y = -60:2:60;
[X, Y] = meshgrid(x, y);
Z = (17200 - X.^2 - 2*Y.^2)/200;
surf(X,Y,Z), axis vis3d equal
A = trapz(x, Z, 2);
V = trapz(y, A)
fun = @(s,t) (17200 - s.^2 - 2*t.^2)/200;
V = integral2(fun, -100, 100, -60, 60)
```



11.8 Initial Value Problems



Example11_08.m: Damped Free Vibrations

[4] This program solves the IVP defined in Eq. (a), last page. →

```
dampedFreeVibration

function dampedFreeVibration
m = 1; c = 1; k = 100; delta = 0.2;
[time, sol] = ode45(@fun, [0,2], [delta, 0]);
plot(time, sol(:,1));
xlabel('Time (sec)')
ylabel('Displacement (m)')
yyaxis right
plot(time, sol(:,2));
ylabel('Velocity (m/s)')
title('Free Vibration of a SDOF Damped System')
grid on

    function ydot = fun(t, y)
        ydot(1) = y(2);
        ydot(2) = (-k*y(1)-c*y(2))/m;
        ydot = ydot';
    end
end
```

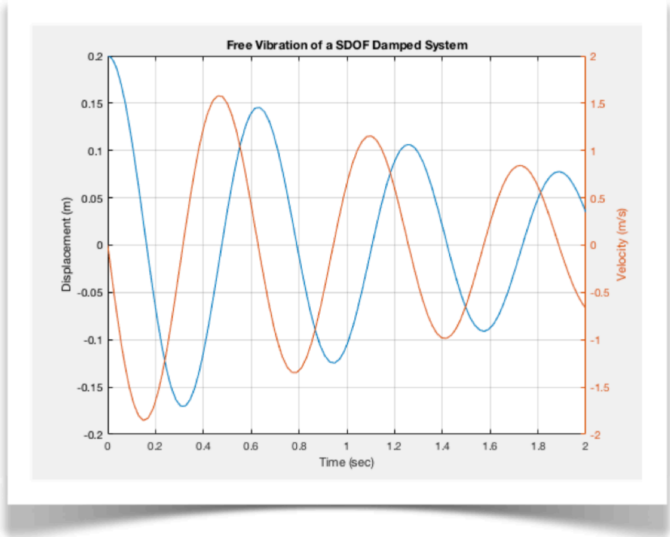


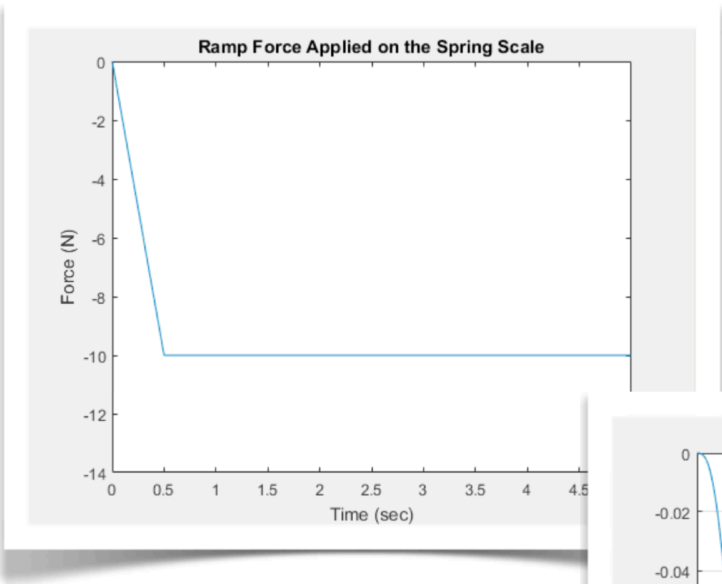
Table 11.8 Summary of Functions for IVPs	
Functions	Description
`[t, sol] = ode45(@fun, tspan, y0)`	Solve nonstiff DEs; medium order method
`[t, sol] = ode15s(@fun, tspan, y0)`	Solve stiff DEs and DAEs; variable order method
`[t, sol] = ode23(@fun, tspan, y0)`	Solve nonstiff DEs; low order method
`[t, sol] = ode113(@fun, tspan, y0)`	Solve nonstiff DEs; variable order method
`[t, sol] = ode23t(@fun, tspan, y0)`	Solve moderately stiff ODEs and DAEs; trapezoidal rule
`[t, sol] = ode23tb(@fun, tspan, y0)`	Solve stiff DEs; low order method
`[t, sol] = ode23s(@fun, tspan, y0)`	Solve stiff DEs; low order method
`[t, sol] = ode15i(@fun, tspan, y0)`	Solve fully implicit DEs, variable order method
`ydot = fun(t, y)`	User-defined function to be input to ODE-IVP solvers
Details and More: *Help>MATLAB>Mathematics>Numerical Integration and Differential Equations>Ordinary Differential Equations*	

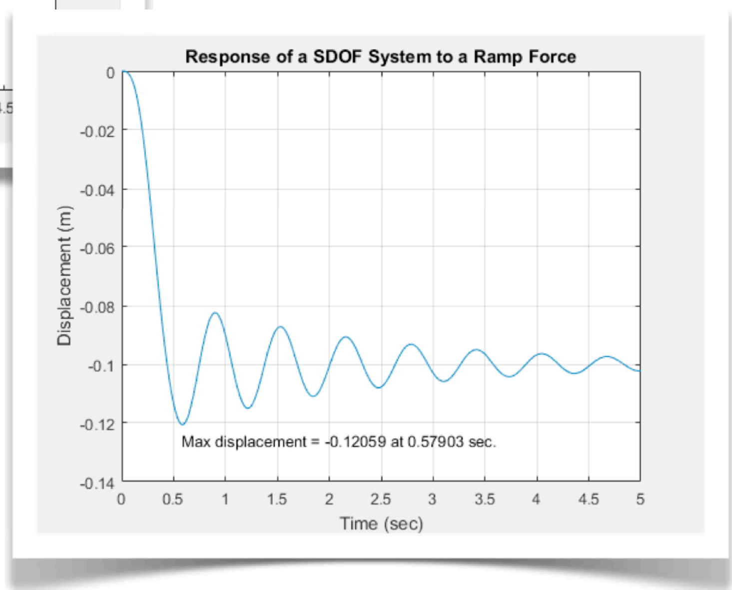
11.9 IVP: Placing Weight on Spring Scale

Example11_09.m: Spring Scale

[2] This program solves the IVP described in Eq. (a). →

```
springScale

function springScale
m = 1; c = 1; k = 100; W = -10; T0 = 0.5;
[time, sol] = ode45(@fun, [0,5], [0,0]);
plot(time,sol(:,1)), grid on
xlabel('Time (sec)'), ylabel('Displacement (m)')
title('Response of a SDOF System to a Ramp Force')
[M, I] = min(sol(:,1));
text(time(I), M-0.005, ['Max displacement = ', ...
    num2str(M), ' at ', num2str(time(I)), ' sec.'])

    function ydot = fun(t, y)
        force = W*t/T0*(t<T0)+W*(t>=T0);
        ydot(1) = y(2);
        ydot(2) = (1/m)*(force - c*y(2) - k*y(1));
        ydot = ydot';
    end
end
```


Ramp Force Applied on the Spring Scale
Force (N)
Time (sec)
0
-2
-4
-6
-8
-10
-12
-14
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5


Response of a SDOF System to a Ramp Force
Displacement (m)
Time (sec)
Max displacement = -0.12059 at 0.57903 sec.
0
-0.02
-0.04
-0.06
-0.08
-0.1
-0.12
-0.14
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

11.10 ODE-BVP: Deflection of Beams

Example11_10.m: Deflection of Beams

[3] This program solves the BVP defined in Eq. (a), last page.

```
beamDeflection

function beamDeflection
w = -0.1; h = 0.1; L = 8; E = 2.1e11; q = 500;
I = w*h^3/12;
solinit = bvpinit(linspace(0,L,1000), [1, 1, 1, 1]);
sol = bvp4c(@odefun, @bcfun, solinit);
x = sol.x; y = sol.y(1,:);
plot(x, y*1000)
xlabel('x (m)'), ylabel('y (mm)')
title('Deflection of Uniformly Loaded and Simply Supported Beam')
[M, I] = min(y);
text(x(I), M*1000, ['Maximum deflection = ', num2str(-M*1000)])

    function yprime = odefun(x, y)
        yprime = zeros(4,1);
        yprime(1) = y(2);
        yprime(2) = y(3);
        yprime(3) = y(4);
        yprime(4) = q/(E*I);
    end

    function residual = bcfun(y0, yL)
        residual = zeros(4,1);
        residual(1) = y0(1);
        residual(2) = y0(3);
        residual(3) = yL(1);
        residual(4) = yL(3);
    end
end
```

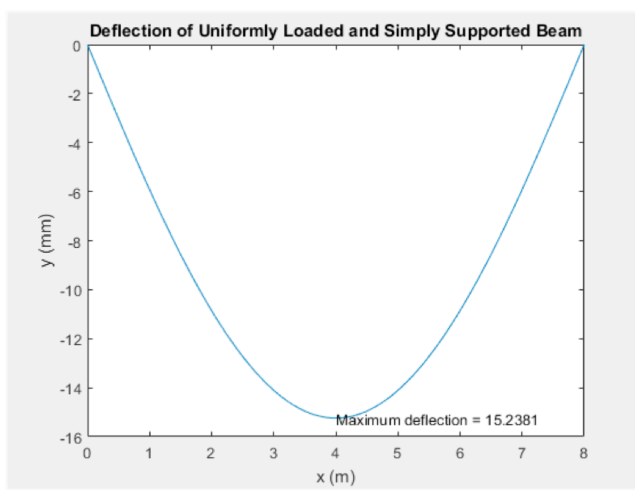


Table 11.10 Summary of Functions for ODE-BVPs

Functions	Description
`sol = bvp4c(@odefun, @bcfun, solinit)`	Solve BVPs for ODEs
`sol = bvp5c(@odefun, @bcfun, solinit)`	Solve BVPs for ODEs
`solinit = bvpinit(x, yinit)`	Form initial guess for BVP solvers
`yprime = odefun(x, y)`	User-defined function defining the ODE
`residual = bcfun(y0, yL)`	User-defined function defining the BCs

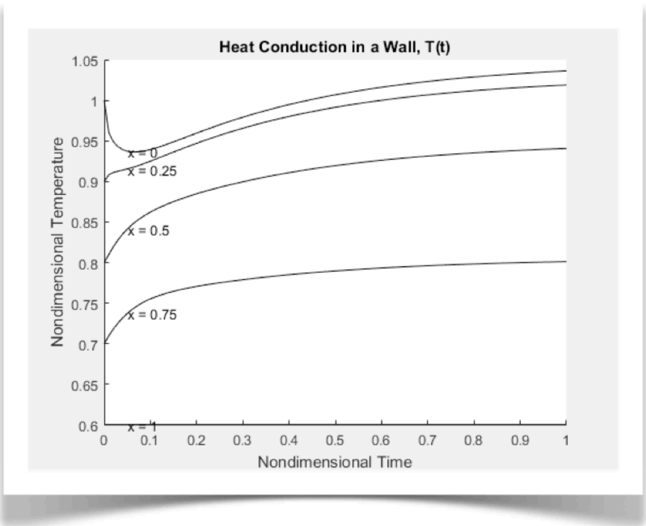
Details and More:
Help>MATLAB>Mathematics>Numerical Integration and Differential Equations>Boundary Value Problems

11.11 IBVP: Heat Conduction in a Wall

Example11_11.m: Heat Conduction in a Wall

[3] This program solves the problem defined in Eqs. (a, b, c), last page. →

```
heatConduction

function heatConduction
A = 0.4; B = 0.05; T1 = 0.6;
x = linspace(0, 1, 21);
t = linspace(0, 1, 101);
T = pdepe(0, @pdefun, @icfun, @bcfun, x, t);
hold on
for k = [1,6,11,16,21]
    plot(t,T(:,k),'k-')
    text(0.05,T(6,k), ['x = ', num2str(x(k))])
end
xlabel('Nondimensional Time')
ylabel('Nondimensional Temperature')
title('Heat Conduction in a Wall, T(t)')

figure, hold on
for k = [1, 51, 101]
    plot(x,T(k,:),'k-')
    text(0.3,T(k,7), ['t = ', num2str(t(k))])
end
xlabel('Nondimensional Distance')
ylabel('Nondimensional Temperature')
title('Heat Conduction in a Wall, T(x)')

    function [c, f, s] = pdefun(x,t,T,Tprime)
        c = 1;
        f = Tprime;
        s = 1;
    end

    function Tinit = icfun(x)
        Tinit = 1-A*x;
    end

    function [pa,qa,pb,qb] = bcfun(a,Ta,b,Tb,t)
        pa = -B*Ta;
        qa = 1;
        pb = Tb-T1;
        qb = 0;
    end
end
```



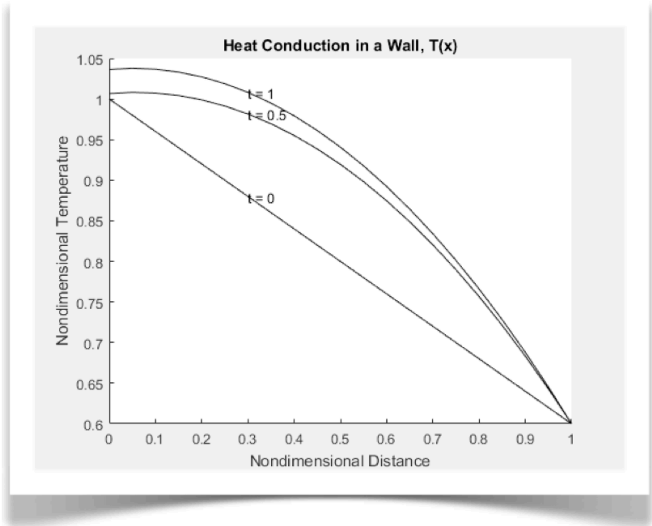


Table 11.11 Summary of Functions for PE-IBVPs

Functions	Description
`sol = pdepe(m, @pdefun, @icfun, @bcfun, x, t)`	Solve a PE-PDE in 1-D
`[c,f,s] = pdefun(x, t, T, Tprime)`	User-defined function defining the PE-PDE
`T0 = icfun(x)`	User-defined function defining the ICs
`[pl,ql,pr,qr] = bcfun(xl,Tl,xr,Tr,t)`	User-defined function defining the BCs

Details and More: Help>MATLAB>Mathematics>Numerical Integration and Differential Equations> Partial Differential Equations>pdepe

Chapter 12

Systems of Nonlinear Equations and Optimization

Applications of optimization techniques are ubiquitous; e.g., curve fitting. Numerical methods for optimization and nonlinear equations are closely related. Many algorithms for solving nonlinear equations are based on minimization of certain functions. For example, in structural mechanics, an equilibrium state is equivalent to a state in which the system has a minimum potential energy. This chapter uses many functions that are part of the Optimization Toolbox. This chapter assumes that you have a license that includes the Optimization Toolbox.

12.1 Nonlinear Equations: Intersection of Two Curves

Example12_01.m: Intersection of Two Curves

[2] This program solves the system of nonlinear equations defined in Eq. (a). →

```
clear, syms x y
fimplicit(x^2/9+y^2/4 == 1, [-4,4]), hold on
fimplicit(y == x^2-4, [-4,4]), axis equal, grid on, box on
legend('x^2/9+y^2/4 = 1', 'y = x^2-4')
title('Intersection of Two Curves')
solveTwoCurves

function solveTwoCurves
xinit = [-1,-1; 1,-1; -1,1; 1, 1];
for k = 1:4
    x = fsolve(@fun, xinit(k,:));
    text(x(1),x(2), ...
        ['x = ', num2str(x(1)), '   y = ', num2str(x(2))], ...
        'HorizontalAlignment', 'center')
end

    function f = fun(x)
        f = zeros(2,1);
        f(1) = x(1)^2/9+x(2)^2/4-1;
        f(2) = x(1)^2-x(2)-4;
    end
end
```

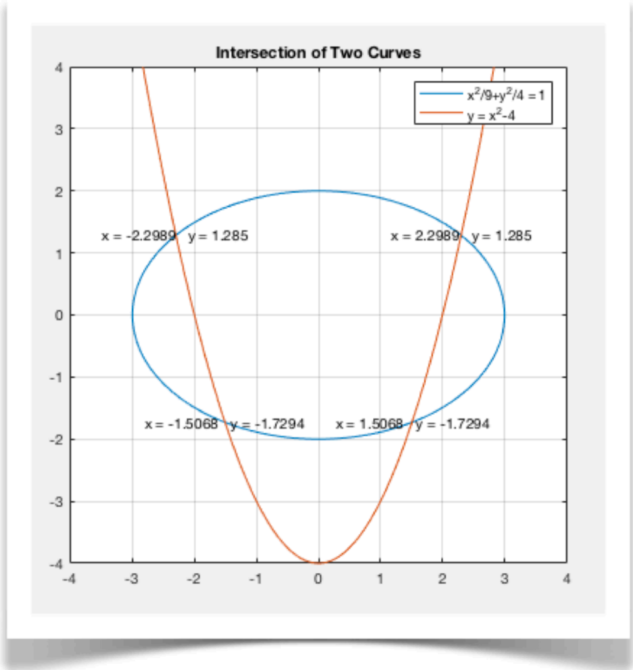
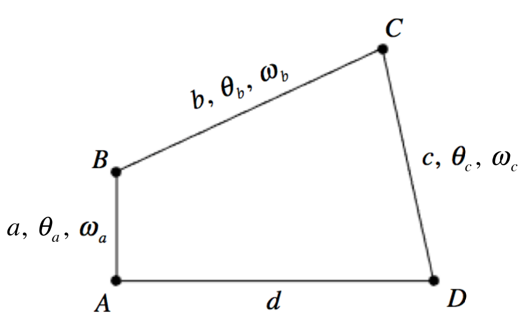
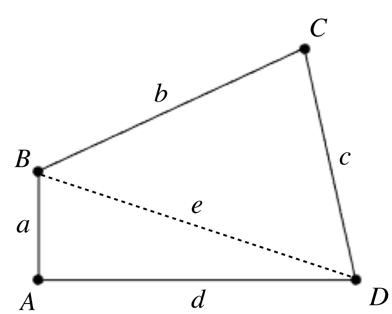


Table 12.1 Summary of Functions for Systems of Nonlinear Equations	
Functions	Description
`x = fsolve(@fun, x0)`	Solve system of nonlinear equations
`f = fun(x)`	User-defined function defining the system of nonlinear equations
Details and More: Help>Optimization Toolbox>Systems of Nonlinear Equations	

12.2 Kinematics of Four-Bar Linkage

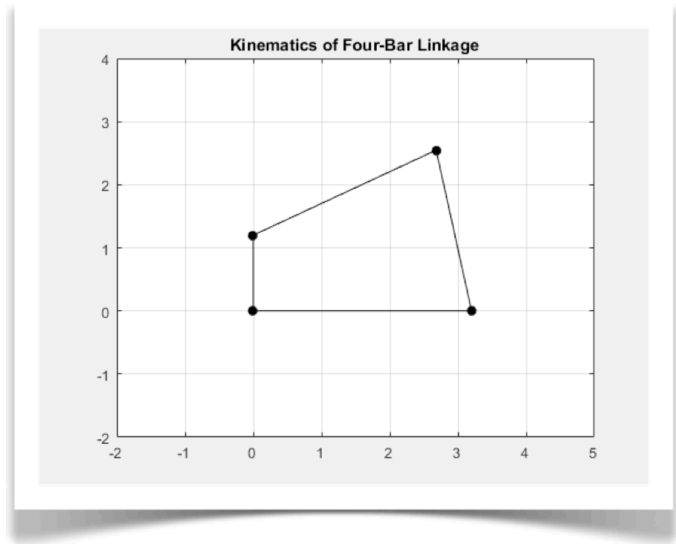




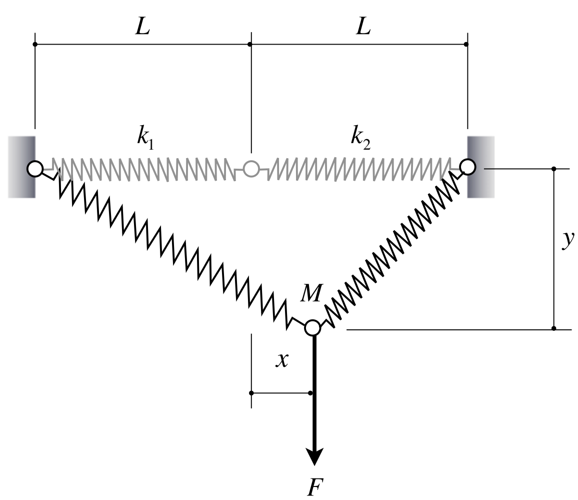
Example12_02.m: Four-Bar Linkage

[2] This program simulates the motion of a four-bar linkage by successively solving Eq. (a), last page. It saves a video in a file (in AVI format), so you can replay the simulation. →

```
fourBarLinkage

function fourBarLinkage
a = 1.2; b = 3; c = 2.6; d = 3.2; omega = 2*pi;
ta = pi/2; e = sqrt(a^2+d^2);
tb = atan(d/a)+acos((b^2+e^2-c^2)/(2*b*e))-pi/2;
tc = tb + acos((b^2+c^2-e^2)/(2*b*c)) - pi;
steps = 50; delta = 2*pi/omega/steps;
x = [0, 0];
options = optimoptions(@fsolve, 'Display', 'off');
for k = 1:steps
    xcoord = [0, a*cos(ta), d-c*cos(2*pi-tc), d, 0];
    ycoord = [0, a*sin(ta), c*sin(2*pi-tc), 0, 0];
    plot(xcoord, ycoord, 'k-o', 'MarkerFaceColor', 'k')
    axis([-2, 5, -2, 4]), grid on
    title('Kinematics of Four-Bar Linkage')
    Frames(k) = getframe;
    x = fsolve(@fun, x, options);
    ta = ta + omega*delta;
    tb = tb + x(1)*delta;
    tc = tc + x(2)*delta;
end
movie(Frames, 5, 30)
videoObj = VideoWriter('Linkage');
open(videoObj);
writeVideo(videoObj, Frames);

    function f = fun(x)
        f = zeros(2,1);
        f(1) = omega*a*sin(ta)+x(1)*b*sin(tb)+x(2)*c*sin(tc);
        f(2) = omega*a*cos(ta)+x(1)*b*cos(tb)+x(2)*c*cos(tc);
    end
end
```


Kinematics of Four-Bar Linkage
4
3
2
1
0
-1
-2
-2
-1
0
1
2
3
4
5

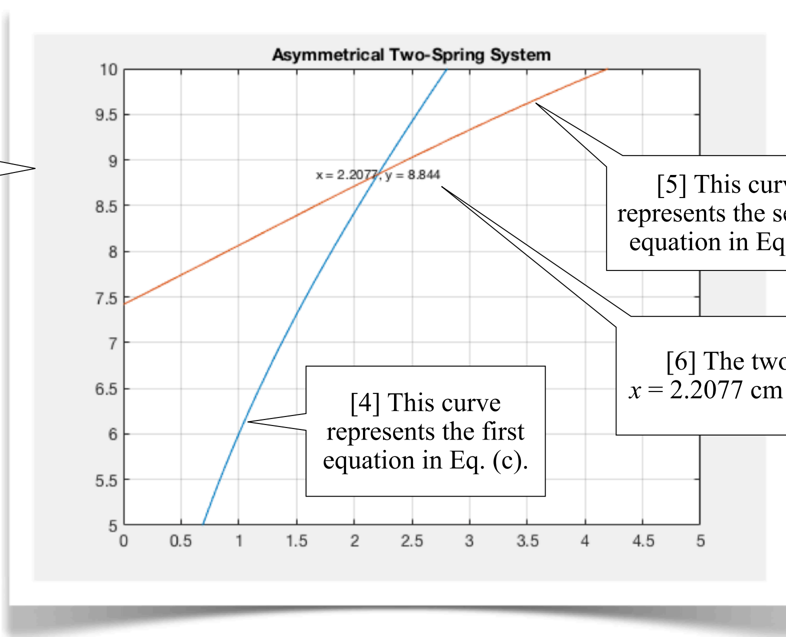
12.3 Asymmetrical Two-Spring System



Example12_03.m: Asymmetrical Two-Spring System

[2] This program solves the problem defined in [1] (last page), producing the graph shown in [3-6].

```
asymmetricalTwoSpring

function asymmetricalTwoSpring
k1 = 1.5; k2 = 6.8; L = 12; F = 9.2;
L1 = @(x,y) sqrt((L+x).^2+y.^2);
L2 = @(x,y) sqrt((L-x).^2+y.^2);
dL1 = @(x,y) L1(x,y)-L;
dL2 = @(x,y) L2(x,y)-L;
fun1 = @(x,y) k1*dL1(x,y).*(L+x)./L1(x,y)-k2*dL2(x,y).*(L-x)./L2(x,y);
fun2 = @(x,y) k1*dL1(x,y).*y./L1(x,y)+k2*dL2(x,y).*y./L2(x,y)-F;
fimplicit(fun1, [0,5,5,10]), grid on, hold on
fimplicit(fun2, [0,5,5,10])
title('Asymmetrical Two-Spring System')

sol = fsolve(@fun, [5,5]);
text(sol(1),sol(2), ...
    ['x = ', num2str(sol(1)), ', y = ', num2str(sol(2))], ...
    'HorizontalAlignment', 'center')

    function f = fun(z)
        f = zeros(2,1);
        f(1) = fun1(z(1), z(2));
        f(2) = fun2(z(1), z(2));
    end
end
```



[3] This is the graphic output of Example12_03.m.

[5] This curve represents the second equation in Eq. (c).

[6] The two curves intersect at $x = 2.2077$ cm and $y = 8.844$ cm. →

[4] This curve represents the first equation in Eq. (c).

12.4 Linear Programming: Diet Problem

	Food	Serving Size	Calorie	Protein	Calcium	Price per Serving
1	Oatmeal	28 g	110 kcal	4 g	2 mg	$0.30
2	Chicken	100 g	205 kcal	32 g	12 mg	$2.40
3	Eggs	2 large	160 kcal	13 g	54 mg	$1.30
4	Whole Milk	237 cc	160 kcal	8 g	285 mg	$0.90
5	Cherry pie	170 g	420 kcal	4 g	22 mg	$2.00
6	Pork and Beans	260 g	260 kcal	14 g	80 mg	$1.90

Reference: http://resources.mpi-inf.mpg.de/conferences/adfocs-03/Slides/Bixby_1.pdf

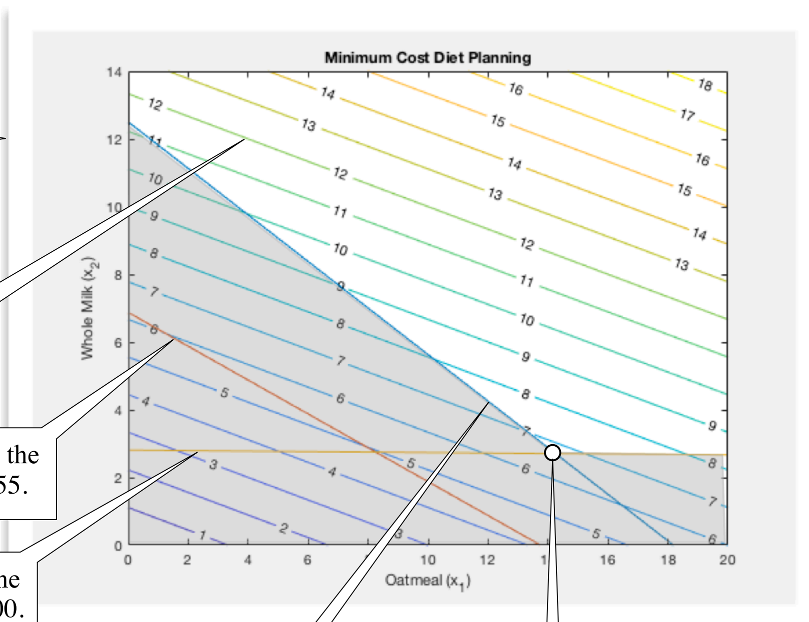
Example12_04.m: Diet Problem

[3] This program solves the linear programming problem defined in [2]. →

```
clear
f = [0.3, 2.4, 1.3, 0.9, 2.0, 1.9];
A = -[110, 205, 160, 160, 420, 260;
        4,  32,  13,   8,   4,  14;
        2,  12,  54, 285,  22,  80];
b = -[2000, 55, 800];
lb = [0 0 0 0 0 0];
[x,fval] = linprog(f,A,b,[],[],lb)

[x1,x4] = meshgrid(0:20, 0:14);
cost = 0.3*x1+0.9*x4;
[C, h] = contour(x1, x4, cost, 0:20); clabel(C, h), hold on
syms x1 x4
fimplicit(110*x1+160*x4==2000, [0,20])
fimplicit(4*x1+8*x4==55, [0,20])
fimplicit(2*x1+285*x4==800, [0,20])
axis([0,20,0,14]), xlabel('Oatmeal (x_1)'), ylabel('Whole Milk (x_2)')
title('Minimum Cost Diet Planning')
```

```
19  Optimal solution found.
20
21  x =
22     14.2443
23           0
24           0
25      2.7071
26           0
27           0
28  fval =
29      6.7096
```

[5] Lines 12-18 produce this graph. It shows contours of the objective function [6] and three inequality constraint functions in the x_1-x_4 space [7-9]. Non-feasible region (area that violates any of constraints, see [7]) is shaded. The shading is manually added by the author.



[6] These parallel lines are contours of the objective function.

[8] This line represents the equation $4x_1 + 8x_4 = 55$.

[9] This line represents the equation $2x_1 + 285x_4 = 800$.

How to Determine the Feasible Region?

[7] This line represents the equation $110x_1 + 160x_4 = 2000$ (x_2, x_3, x_5, and x_6 are zeros), which divides the x_1-x_4 space into two half-spaces. One half-space represents $110x_1 + 160x_4 < 2000$ and the other half-space represents $110x_1 + 160x_4 > 2000$. To determine which half-space represents $110x_1 + 160x_4 > 2000$, one way is to substitute the origin into the left-side of the equation. In this case, since the origin (which is at lower-left) satisfies $110x_1 + 160x_4 < 2000$, the region that satisfies $110x_1 + 160x_4 > 2000$ is the other half-space, i.e., the upper-right. Collection of the points that satisfy all constraints (including bounds) are the feasible region. In this case, the unshaded area is the feasible region (also see [8, 9]).

[10] This is the location where $x_1 = 14.2443$, $x_4 = 2.7071$, at which the value of the objective function is 6.7096, the minimum cost. →

Table 12.4 Summary of `linprog`

Function	Description
`[x,fval] = linprog(f,A,b,Aeq,beq,lb,ub,options)`	Solve linear programming problems
`options = optimoptions(@solver, name, value, ...)`	Create optimization options structure
`options = optimset(name, value, ...)`	Create optimization options structure
Details and More: Help>Optimization Toolbox>Linear Programming and Mixed-Integer Linear Programming	

12.5 Mixed-Integer Linear Programming

Example12_05a.m

[2] This program solves the diet problem with the additional constraints that the servings of the eggs (x_3) and the whole milk (x_4) are integer numbers.

```
clear
f = [0.3, 2.4, 1.3, 0.9, 2.0, 1.9];
A = -[110, 205, 160, 160, 420, 260;
        4,  32,  13,   8,   4,  14;
        2,  12,  54, 285,  22,  80];
b = -[2000, 55, 800];
lb = [0 0 0 0 0 0];
intcon = [3, 4];
[x, fval] = intlinprog(f,intcon, A,b,[],[],lb)
```

```
x =
   13.8182
         0
         0
    3.0000
         0
         0
fval =
    6.8455
```

Object	Weight	Value
1	45 kg	$120
2	30 kg	$90
3	110 kg	$200
4	73 kg	$220
5	20 kg	$100
6	68 kg	$150
7	49 kg	$110
8	150 kg	$400

Example12_05b.m: Binary Integer Programming

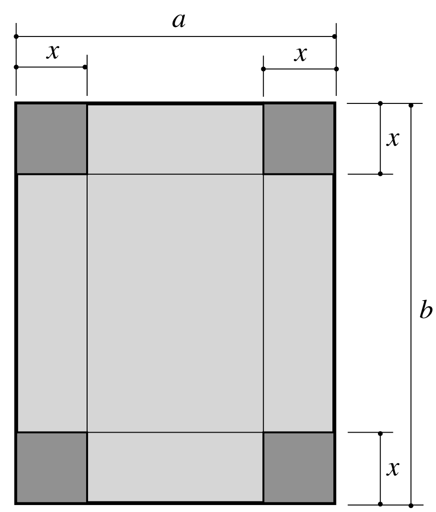
[6] This program solves the problem described in [5].

```
clear
f = -[120,90,200,220,100,150,110,400];
intcon = 1:8;
A = [45,30,110,73,20,68,49,150];
b = [280];
LB = [0 0 0 0 0 0 0 0];
UB = [1 1 1 1 1 1 1 1];
[x, fval] = intlinprog(f, intcon, A, b, [], [], LB, UB)
```

```
x =
         0
    1.0000
         0
    1.0000
    1.0000
         0
         0
    1.0000
fval =
  -810
```

Table 12.5 Summary of Function `intlinprog`	
Function	Description
`[`*`x`*`,`*`fval`*`] = intlinprog(`*`f`*`,`*`intcon`*`,`*`A`*`,`*`b`*`,`*`Aeq`*`,`*`beq`*`,`*`lb`*`,`*`ub`*`,`*`options`*`)`	Mixed-integer linear programming
Details and More: Help>Optimization Toolbox>Linear Programming and Mixed-Integer Linear Programming	

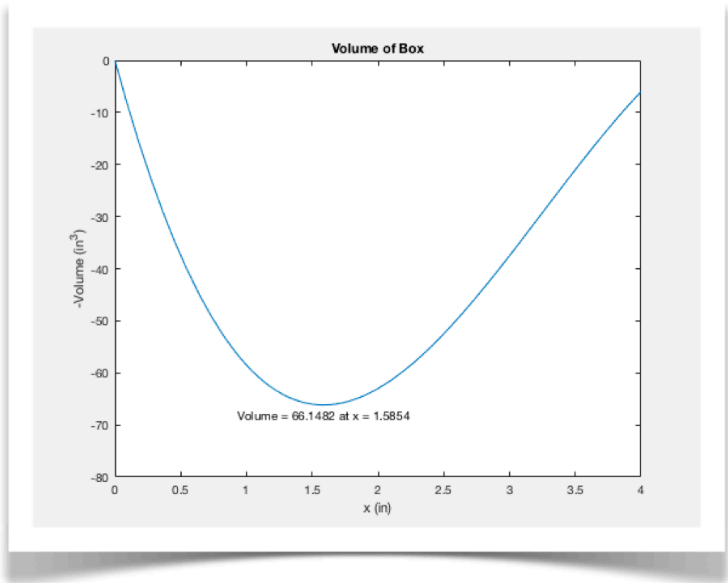
12.6 Unconstrained Single-Variable Optimization

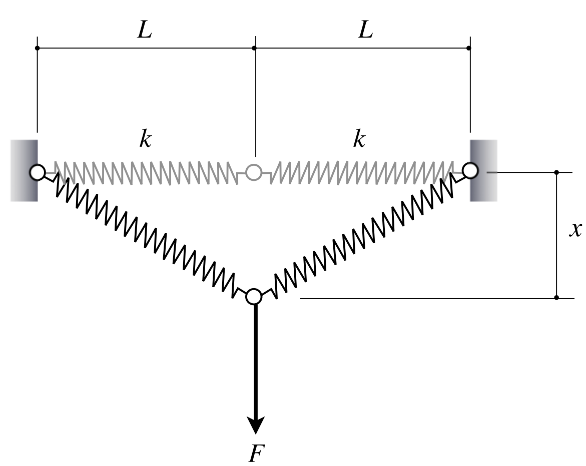


Example12_06a.m: Volume of a Paper Container

[3] This program solves the problem described in [2]. →

```
clear
a = 8.5; b = 11;
objective = @(x) (-x.*(a-2*x).*(b-2*x));
LB = 0; UB = min(a,b)/2;
[x, V] = fminbnd(objective, LB, UB);
fplot(objective, [LB, UB]), axis([0 4 -80 0])
text(x, V-2, ...
    ['Volume = ', num2str(-V), ' at x = ', num2str(x)], ...
    'HorizontalAlignment', 'center')
xlabel('x (in)'), ylabel('-Volume (in^3)')
title('Volume of Box')
```


Volume of Box
-Volume (in^3)
x (in)
Volume = 66.1482 at x = 1.5854
0
-10
-20
-30
-40
-50
-60
-70
-80
0
0.5
1
1.5
2
2.5
3
3.5
4


L
L
k
k
x
F

Example12_06b.m: Symmetrical Two-Spring System

[7] This program solves the symmetrical two-spring system described in [6], last page.

```
clear
k = 6.8; L = 12; F = 9.2;
potential = @(x) (0.5*k*(sqrt(L^2+x.^2)-L).^2*2-F*x);
LB = 0; UB = 10;
[x, PE] = fminbnd(potential, LB, UB);
fplot(potential, [LB, UB]), axis([0 10 -50 0])
text(x, PE-2, ...
    ['Total Potential Energy = ', num2str(PE), ' at x = ', num2str(x)], ...
    'HorizontalAlignment', 'center')
xlabel('x (cm)'), ylabel('Total Potential Energy (N-cm)')
title('Total Potential Energy of Two-Spring System')
```

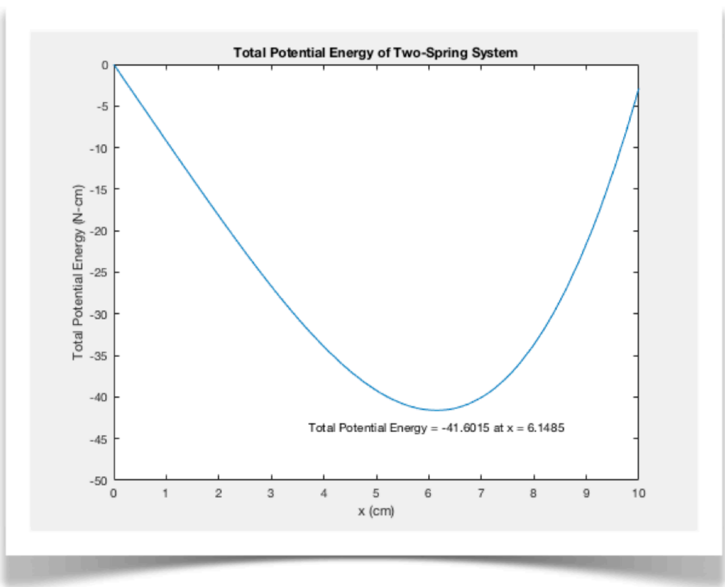
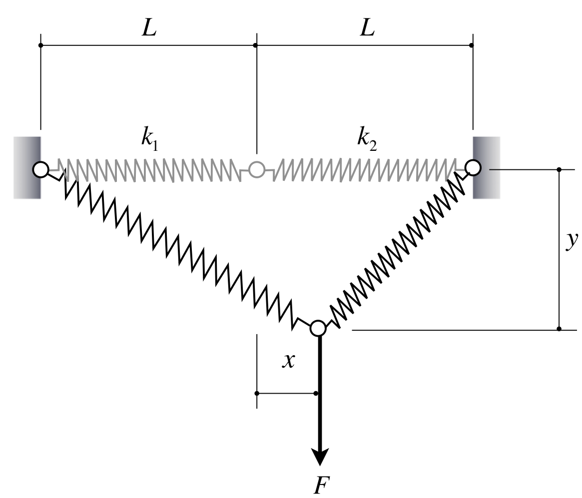


Table 12.6 Summary of `fminbnd`	
Functions	Description
`[`*`x`*`,`*`fval`*`] = fminbnd(@`*`fun`*`,`*`x1`*`,`*`x2`*`,`*`options`*`)`	Find minimum of single-variable function on fixed interval
`f` `=` *`fun`*`(x)`	Function defining the objective function
Details and More: Help>MATLAB>Mathematics>Optimization>Optimizer	

12.7 Unconstrained Multivariate Optimization



Example12_07.m: Asymmetrical Two-Spring System

[3] This program solves the problem described in [2] (last page), producing the graph shown in [4].

```
asymmetricalTwoSpring

function asymmetricalTwoSpring
k1 = 1.5; k2 = 6.8; L = 12; F = 9.2;
L1 = @(x,y) sqrt((L+x).^2+y.^2);
L2 = @(x,y) sqrt((L-x).^2+y.^2);
dL1 = @(x,y) L1(x,y)-L;
dL2 = @(x,y) L2(x,y)-L;
PE = @(x,y) 0.5*k1*dL1(x,y).^2 + 0.5*k2*dL2(x,y).^2 - F*y;
[X, Y] = meshgrid(linspace(0,5), linspace(5,10));
[C, h] = contour(X, Y, PE(X,Y), -60:5:10);
clabel(C, h), xlabel('x'), ylabel('y'), grid on
title('Asymmetrical Two-Spring System')

[sol, fval] = fminunc(@fun, [5,5]);
text(sol(1),sol(2)+0.1, ['fminunc: ', ...
    'x = ', num2str(sol(1)), ', y = ', num2str(sol(2)), ...
    ' PE = ', num2str(fval)], ...
    'HorizontalAlignment', 'center')

[sol, fval] = fminsearch(@fun, [5,5]);
text(sol(1),sol(2)-0.1, ['fminsearch: ', ...
    'x = ', num2str(sol(1)), ', y = ', num2str(sol(2)), ...
    ' PE = ', num2str(fval)], ...
    'HorizontalAlignment', 'center')

    function f = fun(z)
        f = PE(z(1), z(2));
    end
end
```

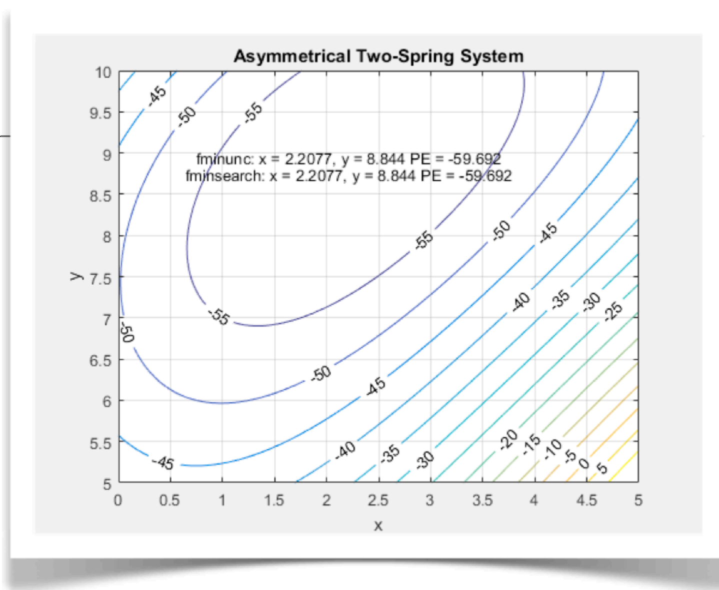


Table 12.7 Summary of `fminunc` and `fminsearch`	
Functions	Description
`[x,fval] = fminunc(@fun,x0,options)`	Find minimum of an unconstrained multivariable function
`[x,fval] = fminsearch(@fun,x0,options)`	Find minimum of an unconstrained multivariable function
`f = fun(x)`	Function defining the objective function
Details and More: Help>Optimization Toolbox>Nonlinear Optimization>Unconstrained Optimization	

12.8 Multivariate Linear Regression

Test no.	x_1 Melt Temperature (Degrees C)	x_2 Injection Speed (% of Full Speed)	x_3 Packing Pressure (kgf/cm^2)	x_4 Mold Temperature (Degrees C)	y Strength (N)
1	150	50	25	30	65.7
2	150	70	50	40	73.7
3	150	90	75	50	70.0
4	170	50	50	50	82.9
5	170	70	75	30	97.3
6	170	90	25	40	106.1
7	190	50	75	40	119.0
8	190	70	25	50	110.3
9	190	90	50	30	111.4

Example12_08.m: Injection-Mold Tests

[2] This program solves the problem described in [1], last page, producing output shown in [3].

```
injectionMoldTest

function injectionMoldTest
X = [1, 150, 50, 25, 30;
     1, 150, 70, 50, 40;
     1, 150, 90, 75, 50;
     1, 170, 50, 50, 50;
     1, 170, 70, 75, 30;
     1, 170, 90, 25, 40;
     1, 190, 50, 75, 40;
     1, 190, 70, 25, 50;
     1, 190, 90, 50, 30];

Y = [65.7, 73.7, 70.0, 82.9, 97.3, 106.1, 119.0, 110.3, 111.4]';

P1 = fminunc(@fun, [0,0,0,0,0]')
P2 = lsqlin(X, Y, [],[])
P3 = X\Y

    function error = fun(P)
        error = norm(X*P-Y);
    end
end
```

```
P1 =
  -98.6164
    1.0942
    0.1658
    0.0280
   -0.1867
P2 =
  -98.6167
    1.0942
    0.1658
    0.0280
   -0.1867
P3 =
  -98.6167
    1.0942
    0.1658
    0.0280
   -0.1867
```

Table 12.8 Summary of Linear Least Squares	
Functions	Description
`p = lsqlin(C,d,A,b,Aeq,beq,lb,ub,options)`	Solve constrained linear least-square problems
`x = A\b (mldivide)`	Solve systems of linear equations **Ax** = **b** for **x**
Details and More: Help>Optimization Toolbox>Least Squares>Linear Least Squares	

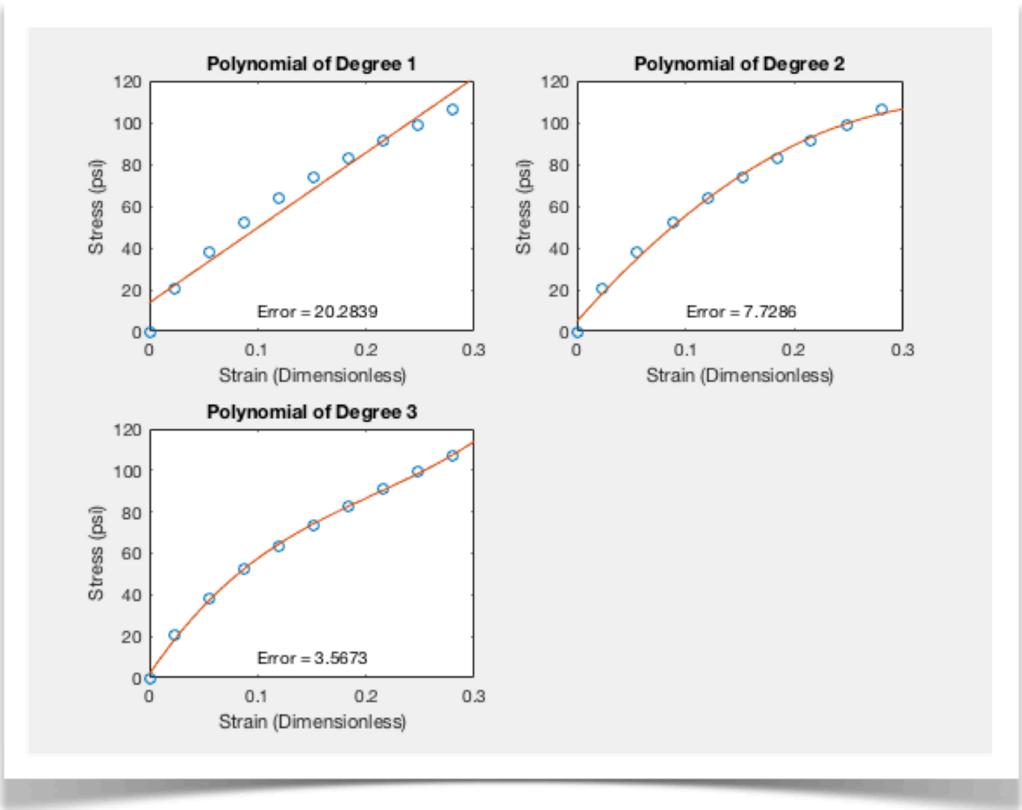
12.9 Non-Polynomial Curve Fitting

Strain (Dimensionless)	0	0.0227	0.0557	0.0880	0.1203	0.1524	0.1840	0.2154	0.2475	0.2797
Stress (psi)	0	20.35	38.29	52.12	63.74	73.77	82.80	91.13	99.07	106.58

Example12_09a.m: Polynomial Curve Fitting

[2] This program fits the data in [1] with polynomials of degrees 1, 2, and 3 and shows the **errors**. →

```
clear
strain = [0,0.0227,0.0557,0.0880,0.1203,0.1524,0.1840,0.2154,0.2475,0.2797];
stress = [0, 20.35, 38.29, 52.12, 63.74, 73.77, 82.80, 91.13, 99.07,106.58];

for k = 1:3
    P = polyfit(strain, stress, k);
    x = linspace(0,0.3);
    y = polyval(P, x);
    subplot(2,2,k)
    plot(strain, stress, 'o', x, y)
    axis([0,0.3,0,120])
    xlabel('Strain (Dimensionless)')
    ylabel('Stress (psi)')
    title(['Polynomial of Degree ', num2str(k)])
    error = norm(stress-polyval(P,strain));
    text(0.1, 10, ['Error = ', num2str(error)])
end
```


Polynomial of Degree 1
Stress (psi)
Strain (Dimensionless)
Error = 20.2839
Polynomial of Degree 2
Stress (psi)
Strain (Dimensionless)
Error = 7.7286
Polynomial of Degree 3
Stress (psi)
Strain (Dimensionless)
Error = 3.5673

Example12_09b.m: Non-Polynomial Curve Fitting

[5] This program fits the data listed in [1] with a function of the form $\sigma = a\varepsilon^b$, where σ is the stress, ε is the strain, and a and b are parameters to be determined. The functions `lsqcurvefit` and `lsqnonlin` (Table 12.9, next page) are used for the non-polynomial curve fittings. Both functions have the same result; the difference is the way they specify the user-defined functions, explained in [8], next page.

```
clear
strain = [0,0.0227,0.0557,0.0880,0.1203,0.1524,0.1840,0.2154,0.2475,0.2797];
stress = [0, 20.35, 38.29, 52.12, 63.74, 73.77, 82.80, 91.13, 99.07,106.58];

fun1 = @(p, xdata) p(1)*xdata.^p(2);
fun2 = @(p) p(1)*strain.^p(2)-stress;
p0 = [0, 0];
[p1, err1] = lsqcurvefit(fun1, p0, strain, stress)
[p2, err2] = lsqnonlin(fun2, p0)
e = linspace(0, 0.3);
s = p1(1)*e.^p1(2);
plot(strain, stress, 'ko', e, s, 'k-')
xlabel('Strain (\epsilon, Dimensionless)'), ylabel('Stress (\sigma, psi)')
text(0.1, 20, ['\sigma = ', num2str(p1(1)), '\times\epsilon^{', ...
    num2str(p1(2)),'}'])
text(0.1, 10, ['Error = ', num2str(err1)])
legend('Experimental Data', 'Fitting Curve', 'Location', 'southeast')
title('Stress-Strain Relationship')
```

```
...
p1 =
  240.4015    0.6326
err1 =
    4.7698
...
p2 =
  240.4015    0.6326
err2 =
    4.7698
...
```

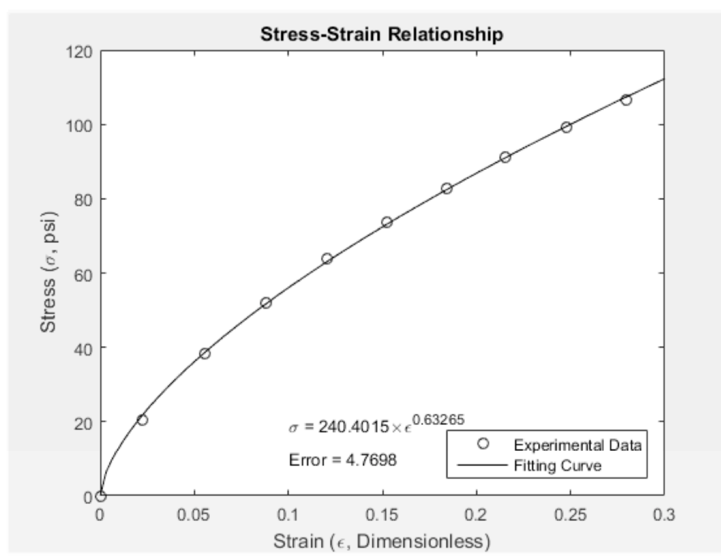
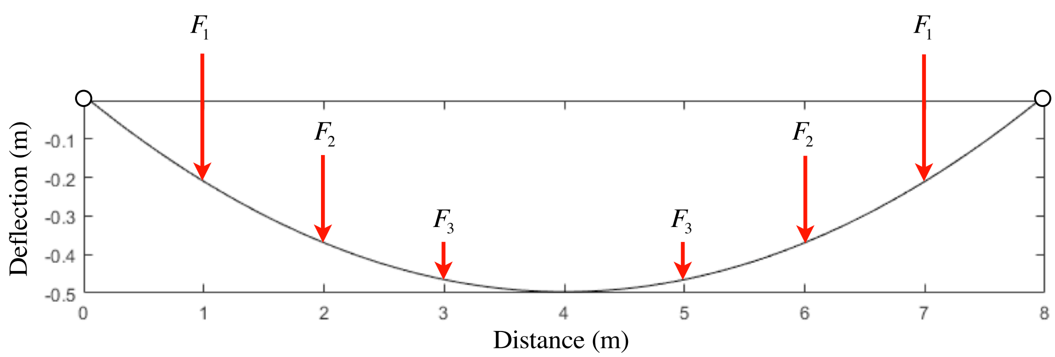


Table 12.9 Summary of Functions `lsqcurvefit` and `lsqnonlin`	
Functions	Description
`[p,err] = lsqcurvefit(@fun1,p0,x,y,lb,ub,options)`	Solve nonlinear curve-fitting problems
`[p,err] = lsqnonlin(@fun2,p0,lb,ub,options)`	Solve nonlinear curve-fitting problems
`f = fun1(p, xdata)`	Function to fit
`f = fun2(p)`	Function whose sum of squares is minimized
Details and More: Help>Optimization Toolbox>Least Squares>Nonlinear Least Squares (Curve Fitting)	

12.10 Constrained Optimization



F_1
F_1
F_2
F_2
F_3
F_3
Deflection (m)
-0.1
-0.2
-0.3
-0.4
-0.5
0
1
2
3
4
5
6
7
8
Distance (m)

Example12_10.m: Parabolically Deflected Beam

[4] This program solves the problem defined in [3], last page. →

```
parabolicBeam

function parabolicBeam
w = 1; h = 0.01; L = 8; E = 210e9; maxSigma = 200e6; deltaCenter = 0.5;
nF = 3; a = L/8*[1,2,3]; F0 = [0,0,0]; LB = [0,0,0]; UB = [inf,inf,inf];
nx = 20; x = linspace(0,L,nx); savedDelta = zeros(1,nx); savedP = [0,0,0];
I = w*h^3/12;
[F, err] = fmincon(@fun, F0, [],[],[],[], LB, UB, @nonlcon)
plot(x, -savedDelta, 'ko', x, -polyval(P, x), 'k-')
axis([0, L, -deltaCenter, 0])
xlabel('Distance (m)'), ylabel('Deflection (m)')
legend('Beam Deflections', 'Fitting Parabola', 'Location', 'north')
title('Parabolically Deflected Beam')

    function err = fun(F)
        delta = zeros(1,nx);
        for k = 1:nF
            delta = delta + deflection(F(k), a(k), x);
            delta = delta + deflection(F(k), L-a(k), x);
        end
        P = polyfit(x, delta, 2);
        err = norm(delta - polyval(P, x));
        savedDelta = delta; savedP = P;
    end

    function [c,ceq] = nonlcon(F)
        sigma = 0; delta = 0;
        for k = 1:nF
            delta = delta + deflection(F(k), a(k), L/2);
            sigma = sigma + stress(F(k), a(k));
            delta = delta + deflection(F(k), L-a(k), L/2);
            sigma = sigma + stress(F(k), L-a(k));
        end
        c(1) = sigma - maxSigma;
        ceq(1) = delta - deltaCenter;
    end

    function delta = deflection(F, a, x)
        R = F/L*(L-a);
        theta = F*a/(6*E*I*L)*(2*L-a)*(L-a);
        delta = theta*x-R*x.^3/(6*E*I)+F/(6*E*I)*((x>a).*((x-a).^3));
    end

    function sigma = stress(F, a)
        M = F*a/2;
        sigma = M*(h/2)/I;
    end
end
```

```
F =
  508.1194  325.1898    0.0144
err =
    0.0210
```

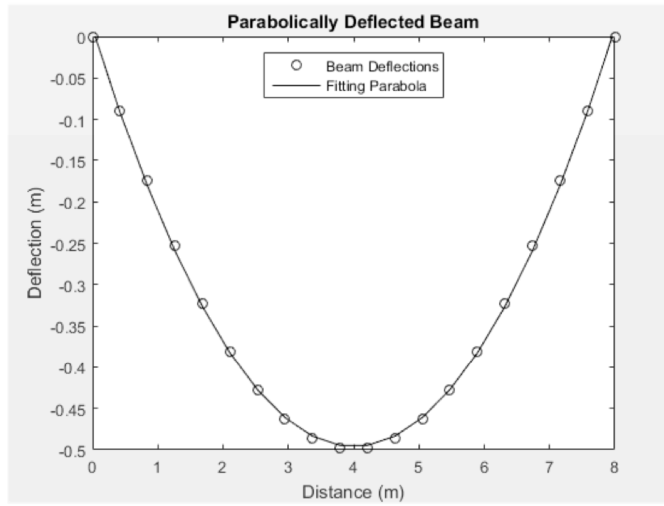


Table 12.10 Summary of `fmincon`

Functions	Description
`[x,fval] = fmincon(@fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)`	Constrained optimization
`f = fun(x)`	Objective function
`[c,ceq] = nonlcon(x)`	Constraints
Details and More: Help>Optimization Toolbox>Nonlinear Optimization>Constrained Optimization	

Chapter 13

Statistics

Statistics is a powerful tool for engineers, but comprehending statistics theories is often challenging for a junior college student. Statistics theories are easier to comprehend by means of statistics experiments, and MATLAB is a perfect tool to conduct statistics experiments, as demonstrated in this chapter. This chapter uses many functions that are part of the Statistics and Machine Learning Toolbox. This chapter assumes that you have a license that includes this toolbox.

13.1 Descriptive Statistics

510.75	536.68	454.82	517.24	506.38	473.85	491.33	506.85	571.57	555.39
473.00	560.70	514.51	498.74	514.29	495.90	497.52	529.79	528.18	528.34
513.43	475.85	514.34	532.60	509.78	520.69	514.54	493.93	505.88	484.25
517.77	477.06	478.62	483.81	441.11	528.77	506.50	484.90	527.41	465.77
497.96	495.17	506.38	506.26	482.70	499.40	496.70	512.55	521.87	522.19
482.73	501.55	475.72	477.73	499.86	530.65	484.61	507.43	495.49	522.35
478.22	500.65	511.05	522.01	530.88	501.72	470.17	485.15	478.77	547.01
487.69	514.96	496.15	517.77	484.70	471.95	471.55	509.76	496.45	496.08
528.39	505.83	503.96	531.75	483.91	513.93	516.70	495.13	504.31	476.68
477.04	502.10	514.45	551.71	486.66	503.75	498.35	461.34	491.22	464.11
516.81	482.24	502.00	489.11	506.07	487.99	509.80	514.79	534.24	496.12
457.23	483.21	527.09	478.56	519.22	502.48	528.73	460.78	496.05	475.84
558.16	516.50	527.58	478.84	490.63	494.55	521.97	494.44	514.03	458.96
492.92	483.53	468.46	510.16	505.64	500.67	473.33	522.55	507.00	494.02
500.46	494.76	465.00	494.29	483.37	480.42	476.87	489.33	459.95	519.28
510.40	499.60	499.30	484.04	520.37	497.34	485.71	527.03	495.50	488.22
494.12	483.04	477.60	550.52	533.11	506.15	474.86	482.69	496.47	515.83
473.36	453.40	471.02	506.67	507.83	509.03	497.39	503.67	490.48	517.24
472.77	509.10	483.03	493.30	511.06	520.78	477.65	525.21	513.20	498.64
496.10	495.65	493.94	500.46	501.03	516.52	530.54	509.34	495.81	512.50

Example13_01a.m: Random Numbers Generation

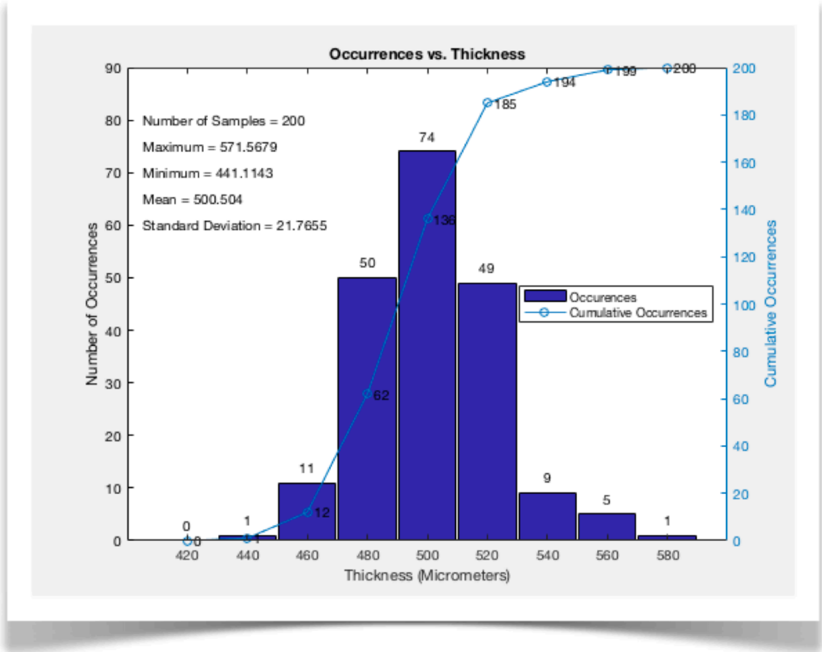
[3] The 200 data in [1] is actually generated by the following commands:

```
clear
rng(0)
data = normrnd(500,20,1,200);
```

Example13_01b.m: Descriptive Statistics

[5] This script uses the data generated in Example13_01a.m, so please execute this script right after the execution of Example13_01a.m. This script introduces some basic terms in Descriptive Statistics, such as **mean** and **standard deviation**, producing a graphic output shown in [6], next page. →

```
mx = max(data)
mn = min(data)
edges = 410:20:590;
counts = histcounts(data, edges);
x = 420:20:580;
bar(x, counts, 0.95)
axis([400,600,0,90])
xlabel('Thickness (Micrometers)'), ylabel('Number of Occurrences')
text(x,counts+3,strsplit(num2str(counts)),'HorizontalAlignment','center')

text(405, 80, ['Number of Samples = ', num2str(length(data))])
text(405, 75, ['Maximum = ', num2str(mx)])
text(405, 70, ['Minimum = ', num2str(mn)])
text(405, 65, ['Mean = ', num2str(mean(data))])
text(405, 60, ['Standard Deviation = ', num2str(std(data))])

cumCounts = cumsum(counts);
yyaxis right
plot(x, cumCounts, 'Marker', 'o')
ylabel('Cumulative Occurrences')
text(x+2, cumCounts, strsplit(num2str(cumCounts)))
legend('Occurences', 'Cumulative Occurrences', 'Location', 'east')
title('Occurrences vs. Thickness')
```

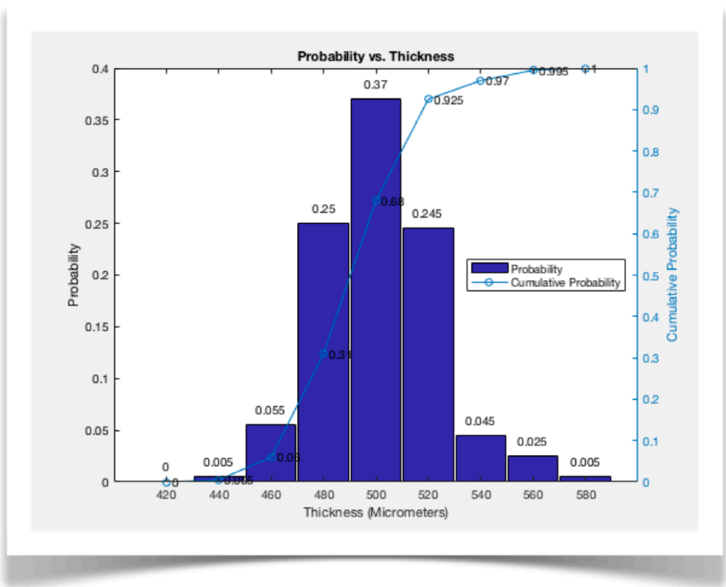


Occurrences vs. Thickness
Number of Samples = 200
Maximum = 571.5679
Minimum = 441.1143
Mean = 500.504
Standard Deviation = 21.7655
Number of Occurrences
Cumulative Occurrences
Thickness (Micrometers)
Occurences
Cumulative Occurrences

Example13_01c.m: Probability

[8] This script uses data generated in Example13_01b.m, so please execute this script right after the execution of Example13_01b.m. This script introduces the terms such as **probability** and **cumulative probability**, producing an graphics output shown in [9].

```
figure
p = counts/length(data);
bar(x, p, 0.95)
xlabel('Thickness (Micrometers)'), ylabel('Probability')
text(x, p+0.015, strsplit(num2str(p)), 'HorizontalAlignment', 'center')

cumP = cumsum(p);
yyaxis right
plot(x, cumP, 'Marker', 'o')
ylabel('Cumulative Probability')
text(x+2, cumP, strsplit(num2str(cumP)))
legend('Probability', 'Cumulative Probability', 'Location', 'east')
title('Probability vs. Thickness')
```



Example13_01d.m: Probability Density

[11] This script uses data generated in Example13_01c.m, so please execute this script right after the execution of Example13_01c.m. This script introduces terms such as **probability density**, producing a graphic output shown in [12].

```
figure
pd = p/20;
bar(x, pd, 0.95)
xlabel('Thickness (Micrometers)'), ylabel('Probability Density')
text(x, pd+0.00075, strsplit(num2str(pd)), 'HorizontalAlignment','center')

cumPD = cumsum(pd)*20;
yyaxis right
plot(x, cumPD, 'Marker', 'o')
ylabel('Cumulative Probability')
text(x+2, cumPD, strsplit(num2str(cumPD)))
legend('Probability Density', 'Cumulative Probability', 'Location','east')
title('Probability Density vs. Thickness')
```

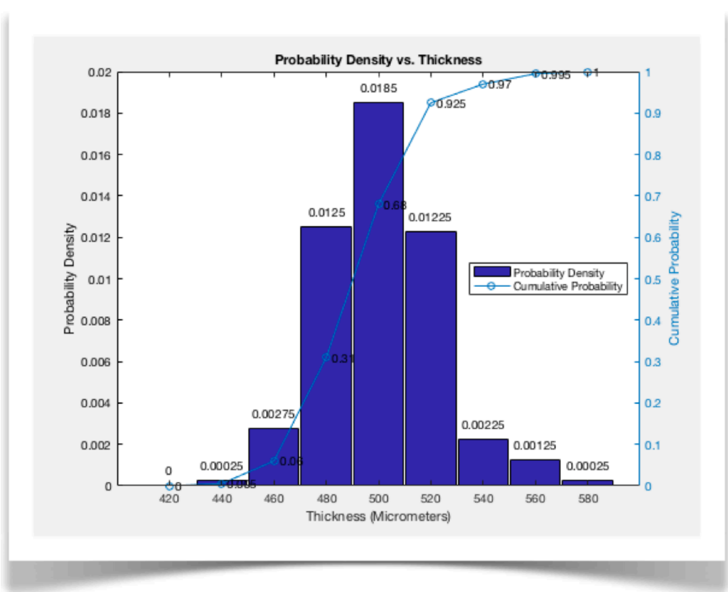


Table 13.1a Random Number Generation	
Functions	Description
`x = rand(row, col)`	Generates random numbers of uniform distribution
`x = randi(imax, row, col)`	Generates random integer numbers of uniform distribution
`x = randn(row, col)`	Generates random numbers of standard normal distribution
`x = normrnd(mu, sigma, row, col)`	Generates random numbers of normal distribution
`rng(seed)`	Controls random number generation
Details and More: Help>MATLAB>Mathematics>Random Number Generation	

Table 13.1b Descriptive Statistics	
Functions	Description
`m = mean(data)`	Average or mean value of an array of numbers
`s = std(data)`	Standard deviation of an array of numbers
`v = var(data)`	Variance of an array of numbers
`counts = histcounts(data, edges)`	Histogram bin counts
`[counts,edges] = histcounts(data,nbins)`	Histogram bin counts
`histogram(data,edges)`	Histogram plot
`histogram(data,nbins)`	Histogram plot
`histfit(data,nbins)`	Histogram with a distribution fit
Details and More: Help>MATLAB>Data Import and Analysis>Descriptive Statistics	

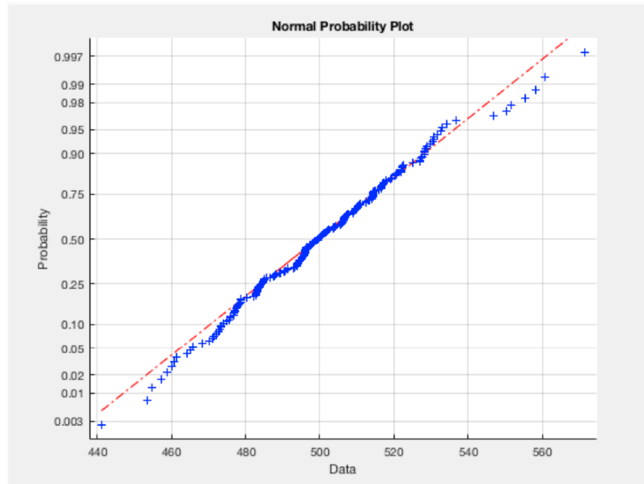
13.2 Normal Distribution

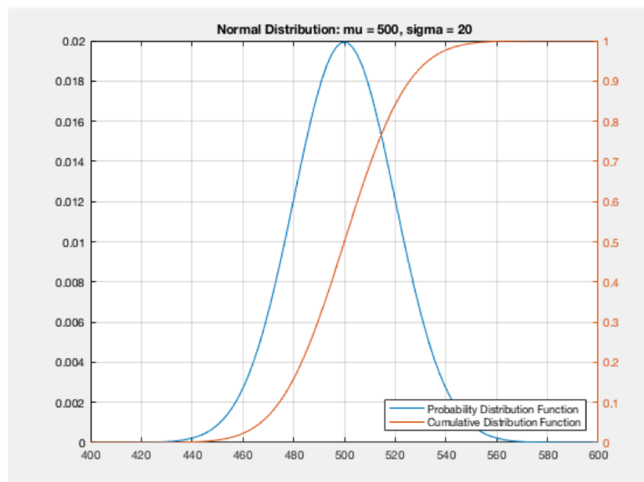
Example13_02a.m: Normal Distribution

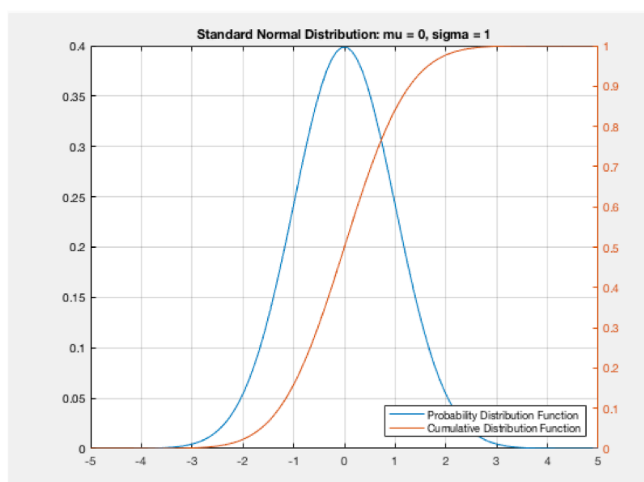
[2] This script produces text output shown in [3] and graphic output shown in [4-6], next page. →

```
clear
rng(0)
mu = 500; sigma = 20; n = 200;
data = normrnd(mu, sigma, 1, n);
normplot(data)
[xbar,s] = normfit(data)

x = linspace(mu-5*sigma, mu+5*sigma);
pdf = normpdf(x, mu, sigma);
cdf = normcdf(x, mu, sigma);
figure
plot(x, pdf)
yyaxis right, plot(x, cdf), grid on
legend('Probability Distribution Function', ...
    'Cumulative Distribution Function', ...
    'Location', 'southeast')
title('Normal Distribution: mu = 500, sigma = 20')
defectRate1 = normcdf(440,mu,sigma) + (1-normcdf(560,mu,sigma))

x = linspace(-5, 5);
pdf = normpdf(x);
cdf = normcdf(x);
figure
plot(x, pdf), x, cdf)
yyaxis right, plot(x, cdf), grid on
legend('Probability Distribution Function', ...
    'Cumulative Distribution Function', ...
    'Location', 'southeast')
title('Standard Normal Distribution: mu = 0, sigma = 1')
defectRate2 = normcdf(-3) + (1-normcdf(3))
```

```
xbar =
  500.5040
s =
   21.7655
defectRate1 =
     0.0027
defectRate2 =
     0.0027
```







Example13_02b.m: Mean and Standard Deviation

[9] This script, producing output shown in [10], confirms the interpretations in [8], last page.

```
clear
mu = 500; sigma = 20;
fun1 = @(x) x.*normpdf(x, mu, sigma);
fun2 = @(x) (x-mu).^2.*normpdf(x, mu, sigma);
mean = integral(fun1, -1000, 1000)
stdev = sqrt(integral(fun2, -1000, 1000))
```

```
mean =
  500.0000
stdev =
   20.0000
```

Table 13.2 Normal Distribution

Functions	Description
`data = normrnd(mu, sigma, row, col)`	Generates random numbers of normal distribution
`normplot(data)`	Generate a normal probability plot
`[xbar,s] = normfit(data)`	Normal parameter estimates
`pd = normpdf(x, mu, sigma)`	Returns probability density of normal distribution
`p = normcdf(x, mu, sigma)`	Returns probability of normal distribution
`x = norminv(p, mu, sigma)`	Returns x-value of normal distribution

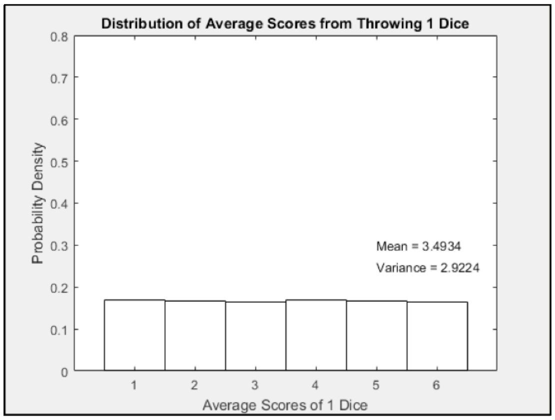
Details and More: Help>Statistics and Machine Learning Toolbox>Probability Distributions>Continuous Distributions>Normal Distribution

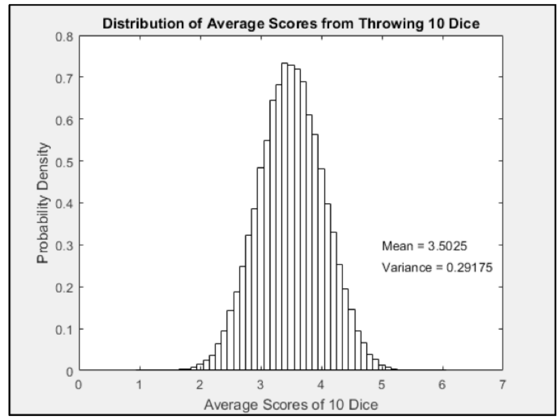
13.3 Central Limit Theory

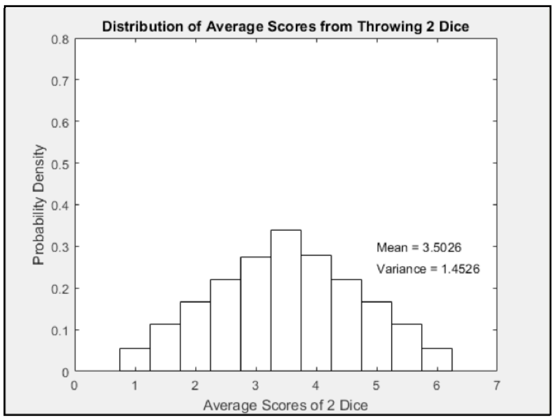
Example13_03a.m: Dice-Throwing Experiments

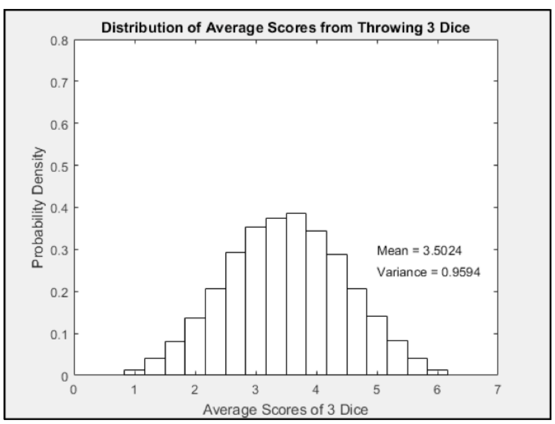
[2] This script simulates the series of experiments described in [1], producing the graphs [3-7], next page. →

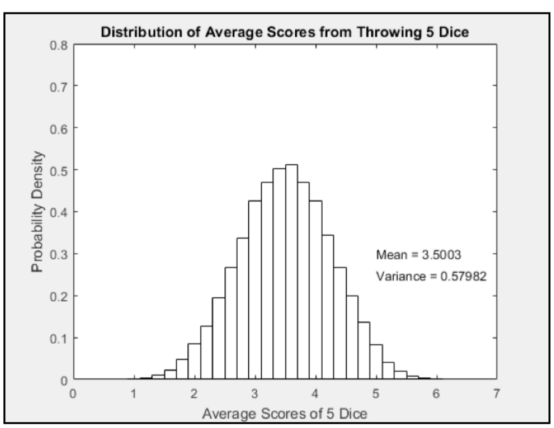
```
clear
n = 50000;
rng(0)
for m = [1,2,3,5,10]
    data = mean(randi(6,m,n),1);
    x = 1:(1/m):6;
    edges = (1-1/m/2):(1/m):(6+1/m/2);
    pd = histcounts(data,edges)/n/(1/m);
    figure
    bar(x, pd, 1.0, 'FaceColor', 'none', 'EdgeColor', 'k')
    axis([0,7,0,0.8])
    text(5, 0.30, ['Mean = ', num2str(mean(data))])
    text(5, 0.25, ['Variance = ', num2str(var(data))])
    xlabel(['Average Scores of ', num2str(m), ' Dice'])
    ylabel('Probability Density')
    title(['Distribution of Average Scores from Throwing ',num2str(m),' Dice'])
end
```


Distribution of Average Scores from Throwing 1 Dice
Probability Density
Average Scores of 1 Dice
Mean = 3.4934
Variance = 2.9224


Distribution of Average Scores from Throwing 10 Dice
Probability Density
Average Scores of 10 Dice
Mean = 3.5025
Variance = 0.29175


Distribution of Average Scores from Throwing 2 Dice
Probability Density
Average Scores of 2 Dice
Mean = 3.5026
Variance = 1.4526

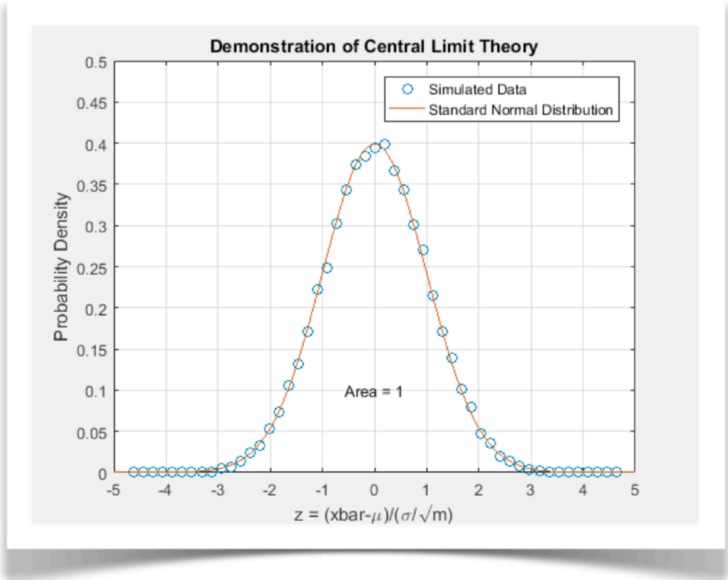

Distribution of Average Scores from Throwing 3 Dice
Probability Density
Average Scores of 3 Dice
Mean = 3.5024
Variance = 0.9594


Distribution of Average Scores from Throwing 5 Dice
Probability Density
Average Scores of 5 Dice
Mean = 3.5003
Variance = 0.57982

Example13_03b.m: Demonstration of Central Limit Theory

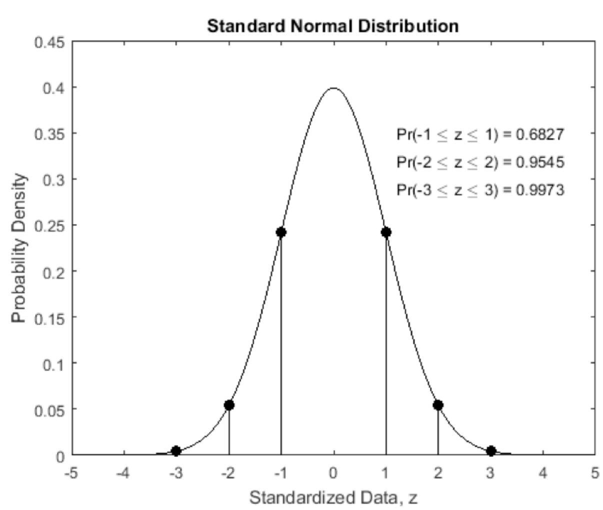
[10] Using the dice-throwing experiments, this script, producing a graphic output shown in [11] (next page), confirms that z, defined in Eq. (b), indeed approaches a standard normal distribution. →

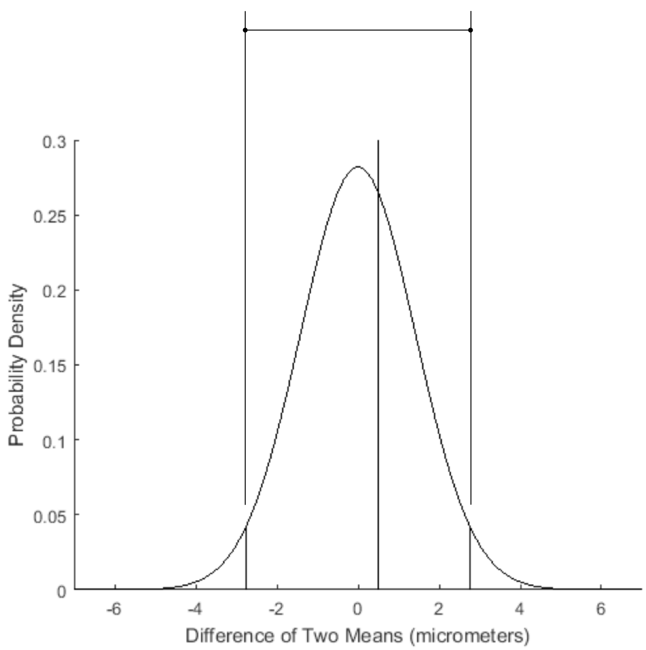
```
clear
n = 50000;
rng(0)
data = randi(6,1,n);
mu = mean(data);
sigma = std(data);

m = 10;
data = mean(randi(6,m,n));
data = (data - mu)/(sigma/sqrt(m));
z = ((1:(1/m):6)-mu)/(sigma/sqrt(m));
edges = (((1-1/m/2):(1/m):(6+1/m/2))-mu)/(sigma/sqrt(m));
pd = histcounts(data,edges)/n/(1/m/(sigma/sqrt(m)));
plot(z,pd,'o'), grid on, hold on
axis([-5,5,0,0.5])
x = linspace(-5,5);
plot(x, normpdf(x))
legend('Simulated Data', 'Standard Normal Distribution')
area = trapz(z,pd);
text(0,0.1,['Area = ', num2str(area)], 'HorizontalAlignment', 'center')
xlabel('z = (xbar-\mu)/(\sigma/\surdm)')
ylabel('Probability Density')
title('Demonstration of Central Limit Theory')
```



Demonstration of Central Limit Theory
Simulated Data
Standard Normal Distribution
Probability Density
Area = 1
z = (xbar-μ)/(σ/√m)
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-5
-4
-3
-2
-1
0
1
2
3
4
5

13.4 Confidence Interval





Example13_04.m: Confidence Interval

[6] This script generates the graph shown in [4, 5]. #

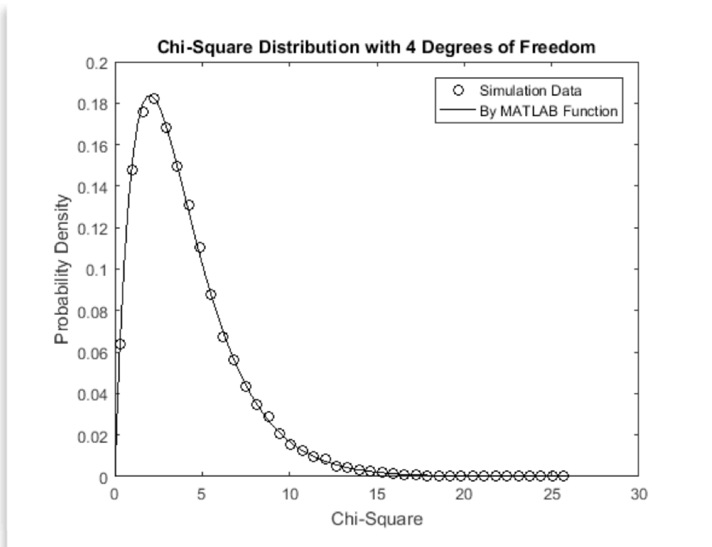
```
clear
x = linspace(-7,7); sigma = sqrt(2);
plot(x,normpdf(x,0,sigma),'k'), hold on, box off
h = gcf; h.Color = 'w';
axis([-7, 7,0,0.3])
z = norminv(0.025,0,sigma); pd = normpdf(z,0,sigma);
plot([z,z], [0,pd], 'k')
z = norminv(0.975,0,sigma); pd = normpdf(z,0,sigma);
plot([z,z], [0,pd], 'k')
z = 0.504;
plot([z,z], [0,0.3], 'k')
xlabel('Difference of Two Means (micrometers)')
ylabel('Probability Density')
```

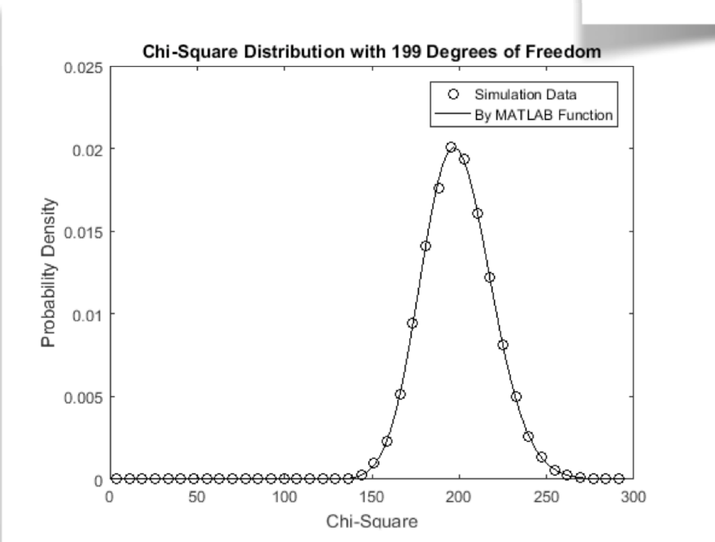
13.5 Chi-Square Distribution

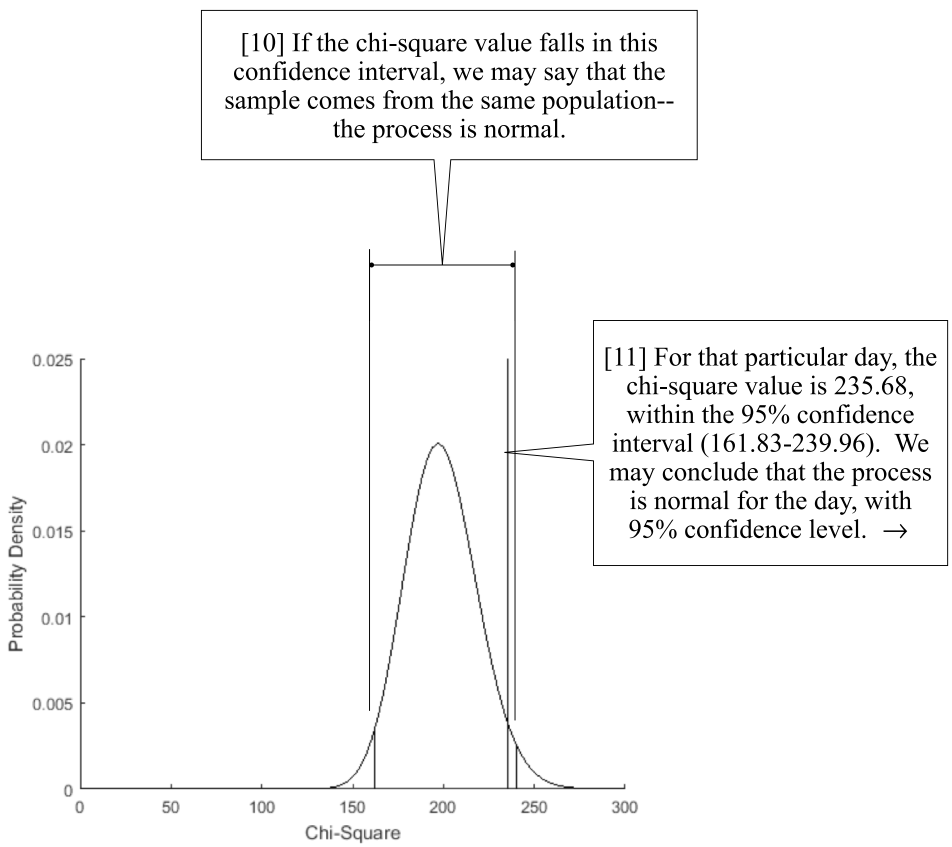
Example13_05a.m: Chi-Square Distribution

[3] This script generates chi-square distributions (see [4-6], next page). If the sample size is 200 (replacing `5` with `200` in line 2, i.e., `m = 200`), the chi-square distributions are shown in [7]. →

```
 1  clear
 2  mu = 500; sigma = 20; m = 5;
 3  n = 50000; rng(0)
 4  data = normrnd(mu, sigma, m, n);
 5  s = std(data);
 6  chi2 = (m-1)*s.^2/sigma^2;
 7  mx = max(chi2); bins = 40;
 8  width = mx/bins;
 9  edges = 0:width:mx;
10  pd = histcounts(chi2, edges)/n/width;
11  x = width/2:width:mx-width/2;
12  plot(x, pd, 'ko'), hold on
13  h = gcf; h.Color = 'w';
14  x = linspace(0,mx);
15  pd = chi2pdf(x, m-1);
16  plot(x, pd, 'k-')
17  xlabel('Chi-Square')
18  ylabel('Probability Density')
19  title(['Chi-Square Distribution with ', ...
20      num2str(m-1), ' Degrees of Freedom'])
21  legend('Simulation Data', 'By MATLAB Function')
```


Chi-Square Distribution with 4 Degrees of Freedom
Simulation Data
By MATLAB Function
Probability Density
Chi-Square
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
5
10
15
20
25
30


Chi-Square Distribution with 199 Degrees of Freedom
Simulation Data
By MATLAB Function
Probability Density
Chi-Square
0.025
0.02
0.015
0.01
0.005
0
0
50
100
150
200
250
300


[10] If the chi-square value falls in this confidence interval, we may say that the sample comes from the same population--the process is normal.
[11] For that particular day, the chi-square value is 235.68, within the 95% confidence interval (161.83-239.96). We may conclude that the process is normal for the day, with 95% confidence level. →
Probability Density
0.025
0.02
0.015
0.01
0.005
0
0
50
100
150
200
250
300
Chi-Square

Example13_05b.m: Chi-Square Test

[12] This script generates a graph shown in [10, 11] and text output shown in [13].

```
clear
s = 21.7655; m = 200; sigma = 20;
chi2 = (m-1)*s^2/sigma^2

x = linspace(0,300);
pd = chi2pdf(x, m-1);
plot(x, pd, 'k-'), hold on, box off
h = gcf; h.Color = 'w';
plot([chi2, chi2], [0, 0.025], 'k-')

lower = chi2inv(0.025, m-1)
plot([lower, lower], [0, chi2pdf(lower,m-1)], 'k-')
upper = chi2inv(0.975, m-1)
plot([upper, upper], [0, chi2pdf(upper,m-1)], 'k-')
xlabel('Chi-Square')
ylabel('Probability Density')
```

```
chi2 =
  235.6842
lower =
  161.8262
upper =
  239.9597
```

Example13_05c.m: Chi-Square Test

[15] This script calculates the significance level (lines 45-50). it also demonstrates the use of the function `vartest` to perform a chi-square test (line 52).

```
44   clear
45   rng(0)
46   mu = 500; sigma = 20; m = 200;
47   data = normrnd(mu, sigma, 1, m);
48   s = std(data);
49   chi2 = (m-1)*s^2/sigma^2;
50   p1 = chi2cdf(chi2, m-1, 'upper')*2
51
52   [h, p2] = vartest(data, sigma^2)
```

```
53   p1 =
54        0.0768
55   h =
56         0
57   p2 =
58        0.0768
```

Table 13.5 Chi-Square Distribution

Functions	Description
pd = chi2pdf(*x*, *dof*)	Returns probability density of chi-square distribution with specified dof
p = chi2cdf(*x*, *dof*)	Returns cumulative probability of chi-square distribution with specified dof
x = chi2inv(*p*, *dof*)	Returns chi-square value, given cumulative probability
[*h*,*p*] = vartest(*data*, *dof*)	Chi-square variance test

Details and More: Help>Statistics and Machine Learning Toolbox>Probability Distributions>Continuous Distributions>Chi-Square Distribution
And: Help>Statistics and Machine Learning Toolbox>Hypothesis Tests

13.6 Student's *t*-Distribution

Example13_06.m: Student's *t*-Distribution

[3] This script generates a graphic output shown in [4, 5], next page. →

```
clear
mu = 500; sigma = 20; m = 5;
n = 50000; rng(0)
data = normrnd(mu, sigma, m, n);
xbar = mean(data);
s = std(data);
t = (xbar-mu)./(s/sqrt(m));

mn = -5; mx = 5; bins = 40;
width = (mx-mn)/bins;
edges = mn:width:mx;
pd = histcounts(t, edges)/n/width;
x = mn+width/2:width:mx-width/2;
plot(x, pd, 'ko'), hold on
h = gcf; h.Color = 'w';

x = linspace(mn,mx);
pd = tpdf(x, m-1);
plot(x, pd, 'k-')

pd = normpdf(x);
plot(x, pd, 'b--')

xlabel('t')
ylabel('Probability Density')
title(['Student''s t-Distribution with ', ...
    num2str(m-1), ' Degrees of Freedom'])
legend('Simulation Data', 'By MATLAB Function')
```

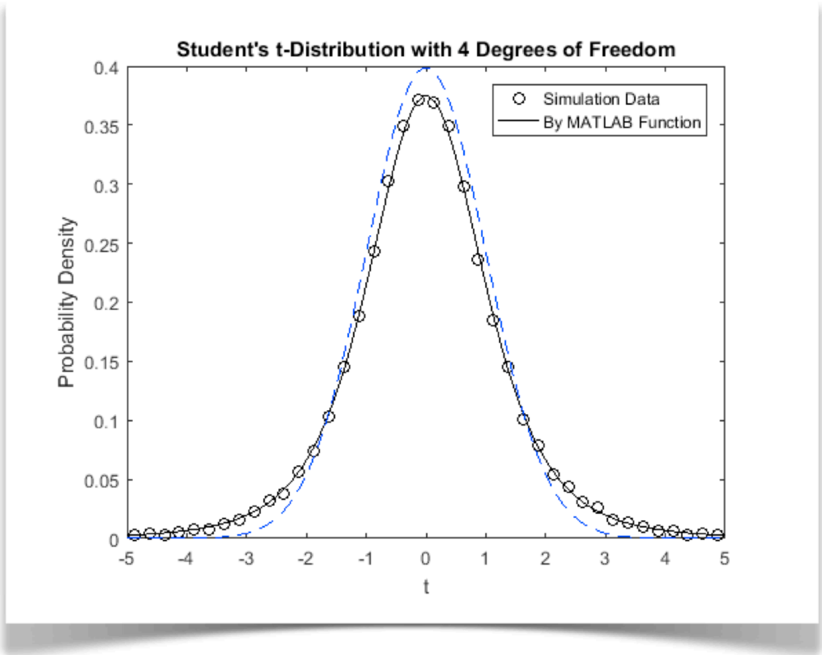


Table 13.6 Student's *t*-Distribution	
Functions	Description
`pd = tpdf(t, dof)`	Returns probability density of *t*-distribution with specified dof
`p = tcdf(t, dof)`	Returns cumulative probability of *t*-distribution with specified dof
`t = tinv(p, dof)`	Returns *t* value, given cumulative probability
Details and More: Help>Statistics and Machine Learning Toolbox>Probability Distributions>Continuous Distributions>Student's t Distribution	

13.7 One-Sample *t*-Test: Voltage of Power Supply

Example13_07.m: Voltage of Power Supply

[2] This script calculates the significance level (lines 2-8). It also demonstrates the use of the function `ttest` to perform the one-sample *t*-test (line 10).

```
 1  clear
 2  mu = 100;
 3  data = [126, 101, 105, 103, 98, 108, 107, 125, 107, 99];
 4  m = length(data);
 5  xbar = mean(data);
 6  s = std(data);
 7  t = (xbar-mu)/(s/sqrt(m))
 8  p1 = tcdf(t, m-1, 'upper')*2
 9
10  [h, p2] = ttest(data, mu)
```

```
11  t =
12      2.5280
13  p1 =
14      0.0323
15  h =
16       1
17  p2 =
18      0.0323
```

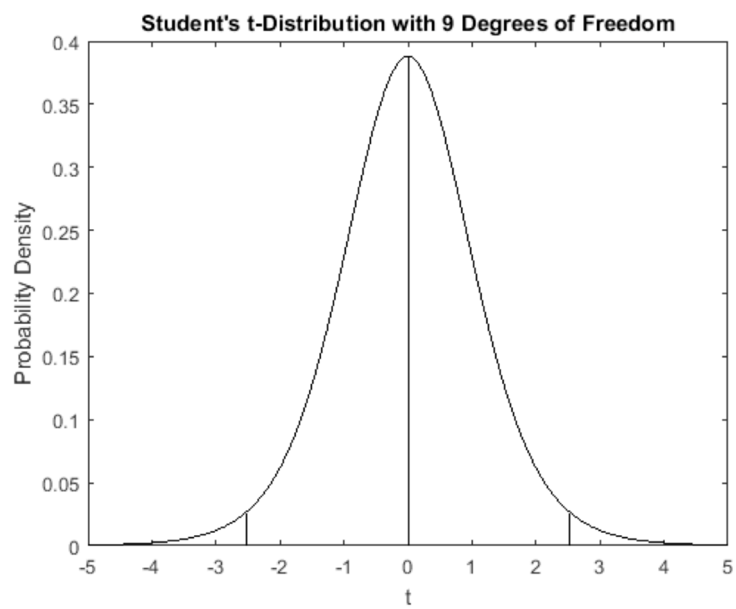


Table 13.7 One-Sample *t*-Test	
Functions	Description
`[h,p] = ttest(data, m)`	One-sample *t*-test
`[h,p] = ttest(data1, data2)`	Paired-sample *t*-test
Details and More: Help>Statistics and Machine Learning Toolbox>Hypothesis Tests	

13.8 Linear Combinations of Random Variables

Example13_08.m: Mean and Variance

[2] This script confirms the properties in [1] by conducting simulations.

```
clear
muX = 10; muY = 20; varX = 1; varY = 2;
n = 50000;
rng(0)
dataX = normrnd(muX, sqrt(varX), 1, n);
dataY = normrnd(muY, sqrt(varY), 1, n);
data = dataX + dataY;
mu1 = mean(data)
var1 = var(data)
data = dataX - dataY;
mu2 = mean(data)
var2 = var(data)
data = dataX*3;
mu3 = mean(data)
var3 = var(data)
data = dataX/3;
mu4 = mean(data)
var4 = var(data)
data = dataX*3 + dataY*2;
mu5 = mean(data)
var5 = var(data)
```

```
mu1 =
   29.9990
var1 =
    2.9874
mu2 =
  -10.0049
var2 =
    2.9795
mu3 =
   29.9911
var3 =
    8.9242
mu4 =
    3.3323
var4 =
    0.1102
mu5 =
   69.9950
var5 =
   16.9154
```

13.9 Two-Sample *t*-Test: Injection Molded Plastic

	Tensile Strength (kgf)								Sample Mean	Sample Variance
Process 1. Without Additive	75	78	65	65	79	77	75	69	72.1	30.7
	74	79	66	67	64	68	76	76		
Process 2. With Additive	79	77	75	70	78	68	71	77	75.1	15.2
	79	74	73	78	72	74	83	74		

Example13_09.m: Injection Molded Plastic

[2] This script calculates the significance level (lines 2-15). It also demonstrates the use of the function `ttest2`, which performs a two-sample *t*-test (line 17), a similar procedure in lines 6-15. →

```
 1  clear
 2  data1 = [75, 78, 65, 65, 79, 77, 75, 69, ...
 3           74, 79, 66, 67, 64, 68, 76, 76];
 4  data2 = [79, 77, 75, 70, 78, 68, 71, 77, ...
 5           79, 74, 73, 78, 82, 74, 83, 74];
 6  m1 = length(data1)
 7  m2 = length(data2)
 8  xbar1 = mean(data1)
 9  xbar2 = mean(data2)
10  var1 = var(data1)
11  var2 = var(data2)
12  varPooled = (var1*(m1-1)+var2*(m2-1))/((m1-1)+(m2-1))
13  varDiffAve = varPooled/m1 + varPooled/m2
14  t = (xbar1-xbar2)/sqrt(varDiffAve)
15  p1 = tcdf(t, (m1-1)+(m2-1))*2
16
17  [h, p2] = ttest2(data1, data2)
```

```
m1 =
    16
m2 =
    16
xbar1 =
   72.0625
xbar2 =
   75.7500
var1 =
   30.7292
var2 =
   17.2667
varPooled =
   23.9979
varDiffAve =
    2.9997
t =
   -2.1291
p1 =
    0.0416
h =
     1
p2 =
    0.0416
```

Table 13.9 Two-Sample t-Test

Functions	Description
`[h,p] = ttest2(data1, data2)`	Two-sample t-test
Details and More: Help>Statistics and Machine Learning Toolbox>Hypothesis Tests	

13.10 *F*-Distribution

Example13_10.m: *F*-Distribution

[2] This script generates *F*-distributions as shown in [3-5], next page. →

```
clear
mu = 500; sigma = 20; m1 = 6; m2 = 16;
n = 50000; rng(0)
data1 = normrnd(mu, sigma, m1, n);
data2 = normrnd(mu, sigma, m2, n);
v1 = var(data1);
v2 = var(data2);
F = v1./v2;

mx = 6; bins = 40;
width = mx/bins;
edges = 0:width:mx;
pd = histcounts(F, edges)/n/width;
x = width/2:width:mx-width/2;
plot(x, pd, 'ko'), hold on
h = gcf; h.Color = 'w';

x = linspace(0,mx);
pd = fpdf(x, m1-1, m2-1);
plot(x, pd, 'k-')

xlabel('F')
ylabel('Probability Density')
title(['F-Distribution with Degrees of Freedom ', ...
    num2str(m1-1), '/', num2str(m2-1)])
legend('Simulation Data', 'By MATLAB Function')
```

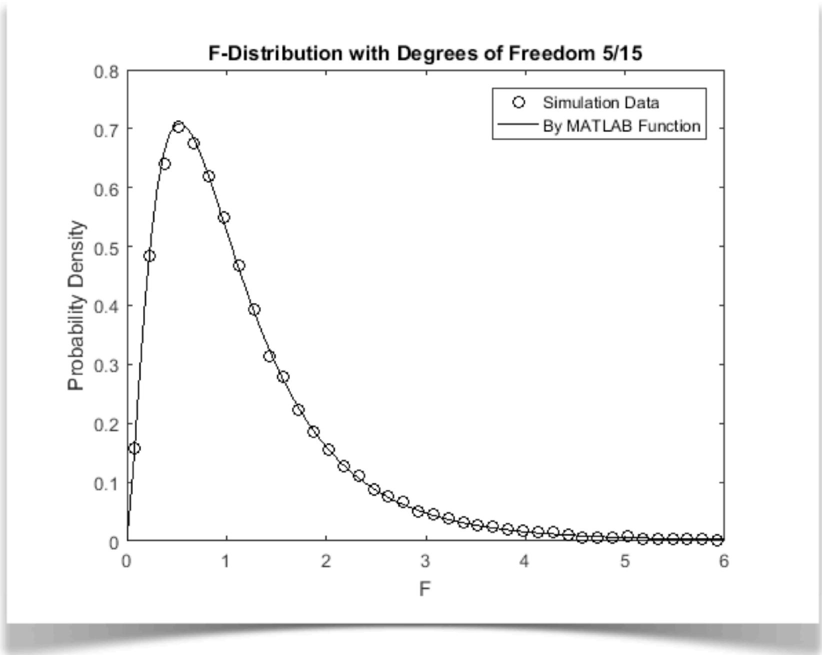


Table 13.10 *F*-Distribution	
Functions	Description
`pd = fpdf(F, dof1, dof2)`	Returns probability density of *F*-distribution
`p = fcdf(F, dof1, dof2)`	Returns cumulative probability of *t*-distribution
`F = finv(p, dof1, dof2)`	Returns *F* value, given cumulative probability
Details and More: *Help>Statistics and Machine Learning Toolbox>Probability Distributions>Continuous Distributions>F Distribution*	

13.11 Two-Sample *F*-Test: Injection Molded Plastic

Example13_11.m: Injection Molded Plastic

[2] This script calculates the significance level (lines 2-10). It also demonstrates the use of the function `vartest2` to perform a two-sample *F*-test (line 12), a similar procedure in lines 6-10.

```
1   clear
2   data1 = [75, 78, 65, 65, 79, 77, 75, 69, ...
3            74, 79, 66, 67, 64, 68, 76, 76];
4   data2 = [79, 77, 75, 70, 78, 68, 71, 77, ...
5            79, 74, 73, 78, 82, 74, 83, 74];
6   m1 = 16; m2 = 16;
7   var1 = var(data1)
8   var2 = var(data2)
9   F = var1/var2
10  p1 = fcdf(F, m1-1, m2-1, 'upper')*2
11
12  [h, p2] = vartest2(data1, data2)
```

```
13  var1 =
14     30.7292
15  var2 =
16     17.2667
17  F =
18      1.7797
19  p1 =
20      0.2755
21  h =
22        0
23  p2 =
24      0.2755
```

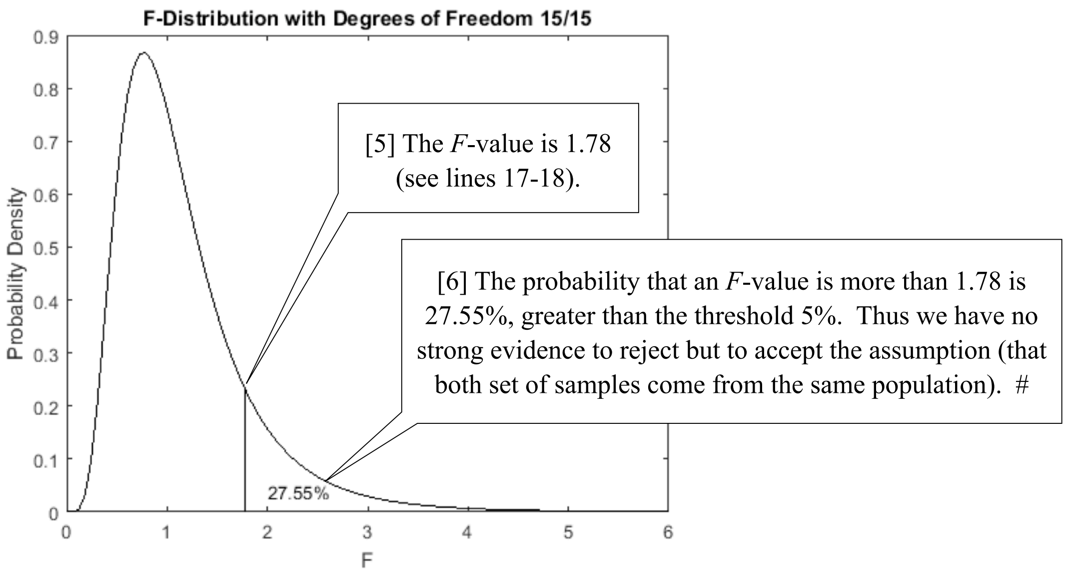


Table 13.11 Two-Sample *F*-Test

Functions	Description
`[h,p] = vartest2(data1, data2)`	Two-sample *F*-test for equal variances
Details and More: Help>Statistics and Machine Learning Toolbox>Hypothesis Tests	

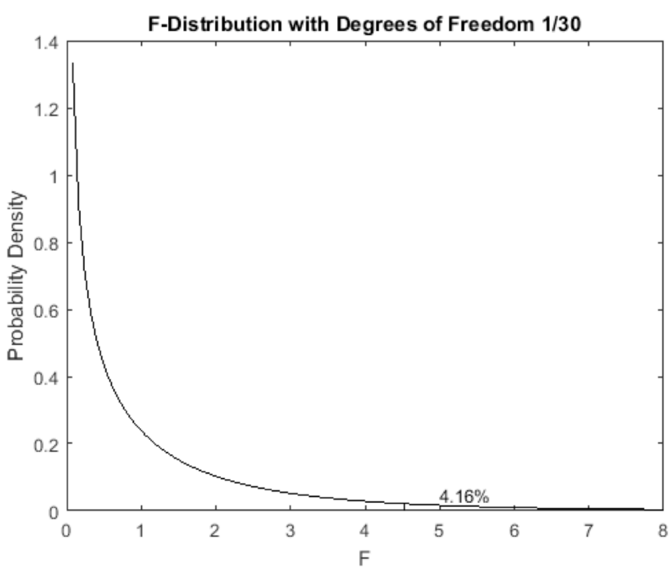
13.12 Comparison of Means by *F*-Test

Example13_12.m: Injection Molded Plastic

[2] This script calculates the significance level using an *F*-distribution.

```
clear
data1 = [75, 78, 65, 65, 79, 77, 75, 69, ...
         74, 79, 66, 67, 64, 68, 76, 76];
data2 = [79, 77, 75, 70, 78, 68, 71, 77, ...
         79, 74, 73, 78, 82, 74, 83, 74];
m = 16;
xbar1 = mean(data1);
xbar2 = mean(data2);
var1 = var(data1);
var2 = var(data2);
varx = (var1*(m-1)+var2*(m-1))/((m-1)+(m-1))
vary = m*var([xbar1, xbar2])
F = vary/varx
p = fcdf(F, 1, (m-1)+(m-1), 'upper')
```

```
varx =
   23.9979
vary =
  108.7812
F =
    4.5329
p =
    0.0416
```


F-Distribution with Degrees of Freedom 1/30
Probability Density
F
4.16%
0
0.2
0.4
0.6
0.8
1
1.2
1.4
0
1
2
3
4
5
6
7
8

Index